IBM

Lotus software

Produced in collaboration with: Redbooks®

Content in this document was produced in collaboration with Lotus® and IBM® Redbooks®.

Redbooks: IBM Lotus Domino Development Best Practices

Robert Baehr
Andre Guirard
Thomas Hampel
Abhishek Jain
Matt Holthe
Shabbir Shaikh

Preface

Important Note about referencing the latest information

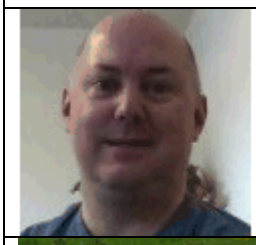
This PDF document represents the overall **Redbooks Wiki for Lotus Domino Development Best Practices.**

In addition to the information contained in this document, also refer to the complete Lotus Redbooks Wiki at the following URL:
http://wiki1.swg.usma.ibm.com/ldd/ddwiki.nsf/

Select "IBM Redbooks Publications" in the navigation. Select "Redbooks Wiki: Domino Development Best Practices" in the navigation. There are tips for navigating the guide on this page.

Meet the Authors

	Robert Baehr (Bob) is the Manager of Business Systems and Development for S.R.I. Quality System Registrar, a pioneer and world leader in Management System Registration. His experience with Lotus Notes and Domino spans two decades (since Release 2), offering software design, system integration, and data integration services to small business through global corporations. Bob was honored to speak at Lotusphere 2009 at the request of the Lotus Foundation's management team. He has been a Lotus Certified Professional Developer, Administrator, and Instructor since Lotus Notes Release 3. Bob currently resides in Cranberry Township, Pennsylvania with his fiance, Cindy.
	Andre Guirard is a longtime Domino developer, former consultant, and for many years a member of the Notes/Domino product development team. He is a frequent presenter at Lotusphere and other conferences, author of many articles on development topics, an OpenNTF contributor, an occasional presence in the LDD forums, and he blogs about Domino best practices.
	Thomas Hampel is an IT Architect at IBM Germany. His key areas of focus are migration projects, performance and delivery service architecture for customers who outsourced to IBM. He is working with Lotus Domino since version 3 and is an IBM Certified Advanced Lotus Developer as well as IBM Certified Advanced Lotus Administrator in various versions up to 8.5. He is also an IBM Certified Advanced Security Professional.

	Abhishek Jain is working with IBM India Software Labs since 2004 and has been in the industry for 10 years. He is currently working as an IT Specialist with IBM Collaboration Services and is skilled on various Lotus products. He is a Certified Lotus Professional, Certified Lotus Administrator, and a Certified Lotus Instructor on both Lotus Domino Administration and Development. Abhishek has been a part of earlier RedWikis Customizing Quickr 8.1 and Best Practices for Building Web Applications for Domino 8.5.1. He has also published developerWorks articles Integrating IBM Lotus Forms with IBM Lotus Domino and Auto-zipping Lotus Notes Attachments.
	**Matt Holthe** is a Certified IT Specialist at IBM in Bloomington, IL. He works as a Field Service Engineer, assisting IBM Premium Support customers with Domino - anything from application assistance, to server upgrades, to new feature enablement. He started working with Lotus Notes version 3.3 in July of 1995. He has been dual certified in Lotus Notes Application Development and Lotus Notes System Administration for versions 5, 6, 6.5, 7, 8, and 8.5.
	Shabbir Shaikh is a System Analyst, and a technology enthusiast working with Lotus Center of Excellence in Capgemini, India. For over 10 years, he is into solution, designing, and development of collaborative applications using Lotus Domino technology for manufacturing, insurance, retail, and banking sector. He has experiences working on process related key roles such as Business Analyst, Offshore Project Coordinator, Project Quality Lead, and Configuration Manager in several engagements. He is an equally enthusiastic trainer to share knowledge on Lotus Application Development best practices by conducting technical training and seminars.

Special thanks to the following people for contributing to this effort

We wish to acknowledge a special thank you to the following sponsors and key stakeholders from the Lotus Forms Development, Product Management, and Lotus IDC Teams:

1. Amanda Bauman - Everyone Writes and IDC Wikis Program Manager
2. Peter Janzen - IBM Senior Product Manager

Additionally, we wish to thank the following members for their technical help and contributions to this wiki:

1. Stephan Wissel - IBM Collaboration and Productivity Advisor , http://www.wissel.net
2. Chris Toohey - Blogger, podcaster, writer, and geek, http://www.dominoguru.com
3. Hunter Medney - IBM Senior I/T Specialist
4. Bruce Lill - Lotus Domino developer and administrator, Kalechi Designs
5. Whei-Jen Chen - IBM Project Leader at the International Technical Support Organization

Chapter 1. Introduction

With Version 8.5.3, IBM® Lotus® Domino® Designer® software further improved its reputation for being a premier collaborative application development tool for Lotus Domino software-based applications. Lotus Domino Designer helps companies improve employee productivity by building and deploying Lotus Domino applications more quickly, thereby, enabling organizations to be more responsive to changing business initiatives. Within this document, we share thoughts and industry best practices about how efficient application development in Lotus Domino can look like. The document discusses certain powerful application development features of Domino XPages development technologies by showing how a developer can apply best practices to efficiently create or extending the functionality of for an existing Domino application. Ideas and concepts presented are meant to be an introduction and do not claim to be a complete list.

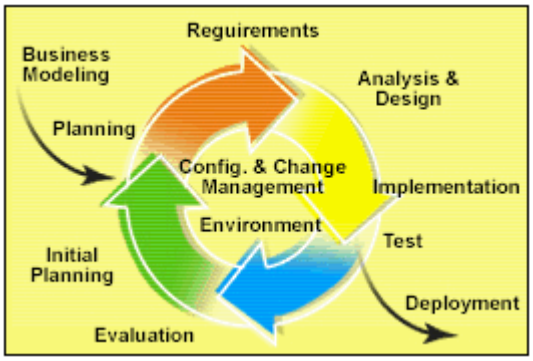
This wiki covers what is needed to build or enhance an application regardless of client and the general practices to make development easier, less time consuming, and easy to support changes within the lifecycle of an application.

Finally, we offer instructions on infrastructure and operational improvements which can benefit performance and maintainability of your environment.

Chapter 2. Application development lifecycle

Application development lifecycle is a commonly used and integral part of application development. It is not a new concept, however, is often under utilized by the developers. In this chapter, we discuss the best practices to manage the Domino application development lifecycle to help you achieve the standardization and stability to the Domino application development.

The following figure shows the phases in application development lifecycle. It is a continue process because the business evolves and the applications are enhanced continuously.



What are the benefits of standardization of Domino application development lifecycle? It helps streamline the different phases of the application development. It weaves all phases through a common thread which leads to logical progression from one phase to another. Builds a strong foundation of requirements on which you establish the pillars of application design. It helps conceptualize and brings focus to several aspects of application design. This actually leads into transformation of the business requirements into an application development.

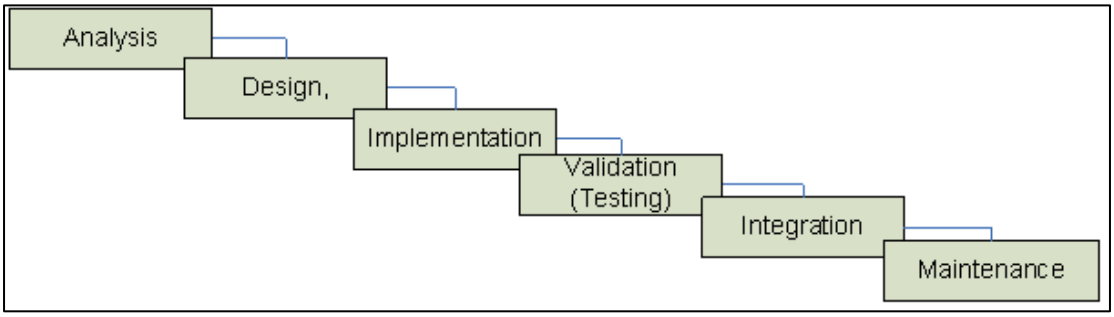
The application development lifecycle further drills down to the actual code generation of the requirements in a standardized manner that brings the application to life. Emphasis on quality aspect of the application been developed that should agree with the expected business requirements. It also provides a platform to stage the application for the user to confirm the application acceptance that agrees with business requirements. Finally, it helps the movement of developed application to make it live in production for its usage by the business.

2.1. Introduction

With a growing number of members in a development team, it is essential to clarify how to structure, plan, and control the development project. Readers of this document might not be new to application development in Domino, here we provide a brief review about the software development project methodology. Following the software development methodology selected by your company is essential for a successful application.

The waterfall method is a commonly used application development methodology. It is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design,

implementation, testing (validation), integration, and maintenance. Each stage is supposed to finish before the next phase starts, or at least have a minimal overlap only. Each phase of the project has to be signed off by the customer and his stakeholders.



While the classic waterfall methodology ends with "maintenance", most projects, including Domino applications, are often extended or upgraded throughout their lifetime. Though each upgrade might be handled as a project by its own, we discuss application upgrade and retirement here together with these application software development phases.

For more information about software development methods, refer to the following:

- DeveloperWorks article *Introducing the IBM Rational Unified Process essentials by analogy*: http://www.ibm.com/developerworks/rational/library/05/wessberg/
- *Crystal Clear* by Alistair Cockburn, ISBN-13: 978-0201699470

2.2. Requirements

In application development lifecycle, requirements literally means: the functional and non-functional needs of the business. These needs become the goals for the application development team. Requirements are the most critical factor in the success of any application development project for one simple reason: it defines what is required by the business to address it needs. It is important to define, communicate, and understand the requirements exactly the way it is supposed to be meant. Any ambiguity wording, misinterpretation, or missing information associated with requirements it results into non-conformance to requirements. Any lapse in the requirement phase that goes undetected can cost you more to rectify this at later stage. It impacts all critical parameters (quality, efforts, and schedule) that define the success or failure of the application development project.

Domino application development generally has a short duration of application development cycle. Considering Domino being a rapid application development platform, it leads to a common issue that the development teams do not allocate sufficient time and attention necessary to manage the requirements appropriately.

In this section we discuss the best approach about requirement management, gathering, and prioritization.

2.2.1 Requirement management

Requirements are discussed and defined during the requirement phase and it continues to refine over the period of application lifecycle. The requirements and its degree of details vary depend on the awareness of the problem and benefit to gain by resolving the problem. The requirements origin is the problem statement, which is the purpose to formulate the requirements to address that problem. Business envisions the benefits of solving an existing problem or altogether innovates its existing process to achieve a competitive advantage. The purpose of requirement management is to organize the requirement, document it, verify that the requirement to ensure it meet the needs and expectation of the business and the stakeholders involved. To understand the requirements, a holistic approach has to be adapted, a proper consideration to be made to identify the functional and non-functional requirements.

A *functional requirement* is the primary functional features that the business expects from the proposed system to perform to address the specific problem. For example:
The product quality application must allow only the users from quality audit team to submit a quality audit report.

The following are some of the common of functional requirement items:

- Data input, output, and processing behavior: Such as the data input and output screens, data format, and data processing.
- Business rules and logic: Such as the task to be performed, data validation, data calculation, and data processes.
- Workflow requirements: Such as the application workflow, level of workflow, document routing, and mailing features.
- Interfaces with other system: Such as payment voucher to be generate using remote function call to a financial database.
- Security requirements: Such as document deletion access is only possible to a specific role.
- Reports or views: The data extraction based on query and representation in desired format.
- User roles and responsibilities: Such as a delegate person has option to make approval decision in absence of a primary approver.
- Error handling: Such as in event of failure to generate the payment voucher, how the order processing should be terminated, and how the log to be generated.

A *non-functional requirement* is the requirement that enhance the functional requirements. Generally, it means the proposed system compliance with the constraints defined by the business. For example:
The new member added to the quality audit team, must be able to use the system within 4 hours of users enrolment in the audit team.

The following are some of the common non-functional requirement items:

- Systems performance constraints: Such as the form load, refresh, or submission time.
- System security constraints: Such as restriction of directory listing, user authorization, data security.
- User interface constraints: Such as GUI screen layout, formatting, and color themes.
- Maintainability: Ease of maintenance of the proposed system. For example, ease of adding a new page to the system, ease to change color themes or static text such as label, ease to add new level of workflow.

- Usability: System usability. For example, system to be accessible using browser client, mobile, thick client.
- Availability: System availability constraints. For example, system to be available 24x7 for the user access.
- Compatibility: System compatibility constraints. For example, cross-browser compatibility, multiple operating system platform.
- Internationalization: System portability to manage users from different locale and time zone. For example, system to support multiple languages and multiple time zone.
- Statutory constraints: The statutory or legal requirements. For example, enforce data access policy, audit trail for edit history.
- Hardware constraints: Systems ability to function within the limit of hardware constrains. For example, system to run at optimum level with 1 GB RAM.
- External Interfaces: System ability to integrate with external sources. For example, interface with Oracle, SAP, and third party tool.
- Operations: Systems compatibility with business operations constraints. For example, possibility for hot or cold system backup.

A good set of requirements have the following characteristics:

- Correct: All requirements should tie back to a customer need.
- Unambiguous: A requirement should have only a single interpretation.
- Complete: All significant needs of users should be addressed.
- Consistent: There should not be any conflict within individual requirements.
- Ranked for importance: Ranking helps the team to address the most critical requirements first and to balance the resources to meet project objectives.
- Verifiable: A person should be able to determine that the developed software meets the requirement.
- Traceable: It should be possible to trace a components requirement to its source .
- Understandable: The user and developer community should be able to comprehend the complete set of individual requirements and the aggregate set implied by the document.

Best practices

Consider the following for requirements management:

- Separate each requirement point as an individual item to maintain a good level of requirement granularity.
- Assign number references to each requirement point and arrange them in the logical order of the flow of requirements as per the dependency on other requirements.
- As the requirement phase progress, keep requirement document updated with the renewed understanding after the interaction and query resolution activities with the stakeholder.
- Each requirement point should focus on a certain aspect. Elaborate each requirement point to a level until it is clear and no ambiguity. If other points come out as a result of elaboration, separate out those individual requirement points.
- Document the requirement in a simple and unambiguous manner, each requirement has to be unique and avoid duplication. It should be testable or verifiable during the testing phase.
- Maintain the requirement document versions regularly to keep the history of requirements evolution from the beginning. This helps to trace the origin of the requirement and every change or refinement done to the requirement. It also helps in maintaining the relationship between the requirements and their solutions.
- Plan a discussion at regular intervals with all the stakeholders to verify that the requirements are in line with the expectations by the business.

    - Identify risks and appropriate mitigation action and update the requirement document accordingly.
- In case of the requirement is complete for the enhancement of an existing system, the appropriate functional and technical impact analysis must be done.
    - Identify the regression test scenario after discussion with business, while you develop the requirement document for such cases.

2.2.2 Requirement gathering

Requirement gathering is a subphase of the requirement phase. In the requirement gathering phase, you identify the stakeholders, the requirement owners, from whom the requirement must be gathered. It is possible to have multiple sources for requirements when the proposed solution spans over more than one workgroup in an organization. You might given a draft requirement document or a simply a problem statement with expected benefits as the initial input from the stakehoders. On the basis of this, you have to gather, elicit, and engage interactions with the requirement owners to refine the raw requirements into meaningful and unambiguous requirement information.

Technology is never a focus during the requirement phase. Assume that a business wants to leverage the existing investment in the Domino infrastructure. They intend to build some capabilities using the rapid application development platform of Domino. Below are some of the requirement gathering phases specific questionnaire for the Domino application development team:

- What are the problems to be solved by the proposed solution?
- What are the historical background of the problems?
- What is the scope of the proposed solution? What are included and excluded from the proposed solution?
- Are there any new problems the proposed solution could cause in the current system?
- Which are the locations the customers use the application? Are there any internationalization requirements?
- What are the security requirements and statutory constraints for the proposed solution?
- What are the access rights for the various users with respect to the application?
- Is there a need for data consistency checks and activity logging?
- What is the initial number of users for the proposed solution?
- What are the possible unknown factors and assumptions?
- What are the standards of performance or other quality attributes, which must be matched or exceeded in the proposed solution?
- What system behaviors do the customer consider as a defect in the system?
- Who are the users and the categories of users if any?
- Are there any technical feasibility issue in addressing a particular requirement? In case of doubt, do a feasibility study or better do prototype to understand business expectation properly.
- Are there any third party components or commercial components to be integrated with Domino application?
- Are there any interfaces to the external data source system, such as RDBMS, web services, Microsoft Office components? What are the expected system response times from external interfaces?
- Is there any data that must be shared across applications?
- What are the hardware limitations that could constrain the design options?

- Are there any compliance constrains for the solution design to adhere some design or programming standards of the client?
- Are rich GUI features required? for example, drag and drop, multiple pop-ups, charts, dynamic data refresh.
- Are there any customer standards or guidelines to be complied with while designing the user interface?
- Will the customer provide screen layouts, graphics, GUI details such as font style, size, and color?
- Does the customer expect a prototype? Does the customer understand that the prototype will be purely navigational and will not be functional?
- Will the user provide the common messages to be thrown up by the application?
    - Will the messages be classified into information, warning, and error messages?
    - Will the information messages be made configurable for the business to manage message text?
- What are the reports requirements the proposed solution has to provide?
    - What are the purpose, intended audience, layout, and the source of data for each report?
    - Are sample report layouts available?
- For each item of input and output, What are the formats of data?
- What are the business rules and validations associated with each user task?
- Is the proposed solution to be available for usage on web browser, notes client, mobile, or a mix of different platforms?
- What are the requirements regarding compatibility? (for example, backwards, cross browser, and so on)
- Are there any constraints around the operating or development environment?
- What are the internationalization and localization requirements?
- Which are the different geographical regions, languages, and currencies that must be supported?
- Input format differences (for example, date, time, and currency) from display formats if any?
- In which format must the system store the information?
- What is the expected peak usage in terms of the maximum number of concurrent users?
- Is the number of users or size of transaction expected to grow in the future?
- What is the volume or size of data to be stored and handled by system?
- What are the intended major features that will probably provide the most value, at least cost, to the largest community of users, and this information to be used in requirement prioritization?

Query register sheet

It is often that the requirements or the problem statement defined by the business is not detailed, lacks clarity, and needs clarifications from the business. The development project team has to allocate sufficient time for clarifying the requirements in the requirement phase. The clarifying action continues, though less frequent, throughout the application development life cycle. The clarifications have an influence on the projects technical and nontechnical decisions.

Your project team should maintain a consolidated *query register sheet* to record the queries, assumptions, and the responses. This query register sheet should be placed at location which is accessible to entire project team. Appoint a single point of contact (SPOC) responsible to ensure the query register is up to date and follows up with the right person to get a response for the open queries. This sheet should include functional

as well as non-functional queries or to record the assumption for confirmation. The query register sheet should be exchanged at regular intervals between the stakeholders. Formulate a format for the query register after discussion within your project team. Below is the sample general attributes of the query register sheet:

- Query description:
  Keep the language simple, non-ambiguous, one logical point at a time, if it helps the respondent, mention the original requirement context.
- Screen dump or exhibit:
  Any additional information such as diagram, flowchart, or screen images of a scenario or representation. Information that helps the respondent to understand the query in a much better way. It helps reduce the query resolution cycle time and the quality of response.
- Severity and impact:
  To understand the risk of delayed response, classify the items in the query register sheet as either high, medium, or low as per the impact or severity of clarification on the requirement.
- Criticality or priority:
  It is possible for your team to generate many queries, considering the limited time on the hands of respondent and dependency of the clarification on the requirement. You could indicate the criticality or priority probably in terms of color code (Yellow: Low | Orange: Medium | Red: High) to help the respondent to focus on the high and medium level of queries first in that order. It allows them to allocate comparatively focus efforts to draft a response with more attention to the details.
- Date raised:
  It is a good practice to track when the query was raised to understand the age of the query and appropriate risk mitigation action to be decided by the project team.
- Status:
  It helps the project team to check on the status of the queries for follow-up and communication with stakeholders, the possible suggested status are Open| Close| Awaiting Inputs| Cancelled| On-Hold
- Assigned to:
  It helps to track the person responsible for the clarification, and ease in follow-up activities.
- Date required:
  Indicating a date by when the response or the clarification is required. The date required section helps communicate the projects expectations of timeline to manage the overall timeline which might be impacted due to the delayed responses.
- Resolution summary and response:
  To capture the responses, the resolution summary, and the clarifications of the queries and the assumptions to trace the requirements and their refinement through the query resolution. These responses help refine the requirement document, removing any ambiguity or misinterpretation.
- Date resolved:
  The resolution date when the queries or assumptions are clarified. It helps in identifying the slippage by comparing the date required and the date resolved.
- Additional comments:
  Any additional comments or overall remarks about the query item in the register.

Best practices

Consider the following when gathering requirements:

- Plan for extensive user involvement to determine correctness of requirements.
- Educate the stakeholders, especially users, about the need for their involvement.
- Expect and plan for the changes in requirements, which are inevitable.

- Ensure to gather both functional and non-functional requirements.
- Understand and use the vocabulary of the domain for better level of discussion with the business user. Include the business terms in the glossary section, which should become part of the requirements documentation.
- Do not think about the solution at the time of requirement gathering, it leads to less focus on requirement and attention to details.
- Make a practice to use flip charts, diagrams, exhibits, prototypes, visual tools, and so on as a starting point and to understand the requirement in better way.
- Focus on understanding business point of view and users tasks rather than user interfaces
- Conduct requirements gathering meetings with the users instead of ad hoc querying them during their regular work.
- In case of remote requirement gathering, use technical aids to record user interactions. Make a point to ask the users' permission before recording the discussion. It helps to trace the original discussion in case there is a change in requirement owners or the members in the project team.
- Ask open-ended questions, it helps to obtain more information from the user's point of view.
- It is a good practice to document the rationale behind each requirement. It helps understand the primary objective of the requirement. It avoids any confusion to choose options which are favorable to this rationale at the time of selecting a solution among several alternatives.
- Try to probe and understand the users' implicit expectations about the proposed system's features, such as performance, usability, efficiency, and reliability. One of the approaches is to ask users what constitute as unacceptable performance, usability, or reliability.
- Do not hesitate to seek clarification for any doubt, unknown term, or ambiguous explanation. Explain to the user about the difficulty to understand the particular business terms been discussed, request for more elaboration.
- You can think of questions that were already asked, rephrase them from a different perspective.
- Maintain proper documentation of all the client's communication for future reference.

2.2.3 Application documentation

The application document is the final requirement document which is the output of requirement gathering and analysis. This document is generally known as the *software requirement specification* or *system requirement specifications*. This document should consist of a complete functional description as well as the behavior of system for all possible business use cases and its interaction with the users. In addition, the application document includes all non-functional requirements that enforce different types of design constrains.

Best practice
When producing the application documentation, consider the following:

- Keep sentences and paragraphs short. Use the active voice.
- Use terms consistently and define them in a glossary or data dictionary.
- To see if a requirement statement is sufficiently well defined, read it from the developer's perspective to learn if it elaborates enough to design and implement.
- Do not merge multiple requirements as a single requirement point.
- It should have clarity about what is in scope and what is not.

- Every function and feature must be described in details with its appropriate non-functional constrains.
- Include use cases with appropriate use case diagram.
- Mention clearly the user acceptance criteria or the acceptance test scenarios for verifying the functions and feature and other non-functional constrains. This helps understand the benchmark expectation of the system for successful acceptance of the solution.

2.2.4 Requirement prioritization

Requirement prioritization is a must when there are multiple requirements but not all of them cab be implemented at the same time due to certain constraints. The constraints can be time, budget, development dependencies, as well as business and technical risks. During project development, there could be new requirements for statutory compliance or change in policy implementation that takes priority on other matters. The constraints from the development project team can be too aggressive schedule to manage all of the requirements, skills availability in development team, and so on. When prioritizing the requirements, it is important to arrive at a mutually agreed terms of all parties that could mean reducing scope or breaking the application into several releases. A proper requirement prioritization helps avoiding wastages (time, effort, money, and other resources) and delivering the application features and functions in time for the business needs.

Both business users and developers should work together in prioritizing the requirements. In the process, derive from the requirements a list of the high level features that are expected from the proposed solution. Assign the importance levels of high, medium, or low to each features. The business users are to determine the level of priority for each feature. The developers are to identify the level of complexity of each feature. The complexity indicates the difficult level (in terms of efforts) to deliver a particular feature. The developers should also identify the level of risk for each feature in high, medium, and low based on the impact on the system had the feature is not delivered in the release under consideration.

Best practice
Consider the following for the requirement prioritization:

- Identify the need for the prioritization of requirement.
- Prioritizing the requirements as early as possible to ensure the resources are invested for critical activities.
- It is essential for the user community and the development team to come to a common agreement on the prioritization.
- Obtain a user sign-off on the requirement prioritization sheet.

2.3. Design

Domino applications are designed to enable users to collaboratively complete a particular process of a business in a group. For example, product development application for manufacturing industry involves several stages, gates, and activities. These activities are to be performed by several workgroups with each having different level of access and constraints in the process flow. This section provides an overview of design principles for developing Domino applications and suggests ways in which you can apply these principles when you create Domino applications.

Designing a Domino application begins with a business problem in backdrop of the requirement specification document generated through the requirement phase. Then work toward designing a solution that provides a positive user experience and maximizes productivity. The following are certain best practices of obtaining a quality design for Domino applications:

- A design demonstrates an architectural structure of the application to be developed.
- A design has to be modular such that the associated features are coupled as a single unit and logical distinct modules are created.
- A design demonstrates the inter dependency between the modules and the constraints, if any.
- A design describes data structure of the respective modules and the data flow between the modules.
- A design contains the details of interfaces (external and internal) with other systems to simplify the complexity of integration with other system.
- A designer aim of a flexible yet robust design structure for the application to easily adapt to the future improvements during the course of development and lifecycle of the application.
- A design describes the workflow process of the domino application and the details specific the workflow.
- A design contains the details of different roles and the access rights in different modules, and stages of workflow.
- A design describes any limitation, constraints, assumption, or special consideration about the application being developed.

2.3.1 Design specification

The design specification is no less than a bible for the application developer to create an application. The design specification translates the business requirements into the building blocks which would eventually bring life to the application. The design specification has cross references to the requirement document to indicate the application design that satisfies all the requirements. The design specification document should include the following sections:

- Graphic user interface (GUI) design
- Database design
- Module level design
- Workflow process
- Program specifications
- Security procedure design
- Detailed design document

GUI design

The graphic user interface (GUI) section describes the details about the GUI of the application such as screen layouts of different modules and their user interface description. It contains the details about the color themes, font settings, and formats for different components. This GUI design specification specifies the GUI standards to be adhered for the application development by the development team. The user accessibility features include the accelerator keys for action buttons and label fields. This section also should include the special GUI considerations if the multiple platforms portability is required, for example, mobile, web browser, and smart devices.

When designing the GUI for Domino applications, consider the following:

- Always use the CSS or Themes to format the user interface and its layout. Avoid inline styles or markup text for formatting the screen layout and user interface elements.
  This would help implement standard look and feel for your application. It is also a good practice for application user interface maintainability. In case the user interface needs to be revamped, you only need to modify the CSS.
- Keep the user interface layout simple and uncluttered. If required, divide it into the logical sections, for example, navigation, actions, and application specific sub-sections.
- Avoid using pop-up unless really required to prevent errors caused by the pop-up blocker that does not allow the pop-ups to be launched. Instead, use the div layers to disable the main user interface screen and use the other div layer as top most with the fields and buttons to get the inputs as applicable.
- Avoid using the large size of images to mitigate problem that might caused by the disk space constraint (See 6.7 Graphics for details).
- Design your GUI with a focus on the target audience device screen resolution for web browser, handheld device, or smartphone.
- Either you set a standard for your application users that the application would be best viewed in particular screen resolution, or design the GUI to support few standard screen resolutions (for example, 800 x 600, 1024 x 768, or 1280 x 1024) for browsers and set similar standards for the smart devices. This standardization for screen resolution or screen size in your application helps to deliver standard GUI to the users.

Database design

This section describes the high-level overview of how the functionality and responsibilities of the modules are structured and integrated with the various modules. The detail of the module is not described in this section but only the overview the association among the modules. The main purpose of this section is to describe the general understanding of how and why the modules are separated, how the individual components work together as a unit to provide the desired functionality from the application.

The description includes the major functionalities and the various capability of the application in terms of features to be provided. Describe the interrelationship between the different modules and the hierarchy of modules and sub-modules. This section mentions the basis of deciding the design points about the modules along with the approaches that have been rejected and the reasons. In this section, the high-level design model diagram for the application is included.

Many Domino applications consist of a single nsf file. For more complex applications, there might be multiple nsf files which work together to make what appears to the end user to be a single application. At the database design stage, identify the number of databases required. Each separate database in the application should represent a logical entity. For example, the product development application of a manufacturing organization might have the following different databases:

- A master database to store all information related to the product, users, and other configuration parameters
- A product database to manage product development lifecycle
- An error log database to track various errors and background processes of application and so on.
  Below is a checklist (not exhaustive) of the factors that lead to the different databases required in a Domino application:

- Complexity of the business processes and interdependencies between the modules. If there seems to be a natural division of the data according to who will use it most and who will maintain it, and if the complexity of the interface between the different parts is low, that suggests separate databases migh make sense.
- If there are several smaller modules which are not inherently complex such as leave management, then a single database is sufficient for the effective functioning of the application.
- The potential number of documents and rate at which the database expected to grow. How many documents are "a lot" varies depending on your hardware and the number of users. But if you expect to be creating thousands of documents a month, you might consider having one or more archive databases, both for better performance and to make it easier for users to create local replicas of just relevant/recent content or search for just desired documents.
- If you think the application will contain a lot of large file attachments, consider a separate attachment repository. Attachments don't have a lot of effect on performance -- number of documents is a much more important factor than total disk space. But users may like to have local replicas, or you might like to create more server replicas, of just the document data, and have just a few server replicas of the attachments, if they are lesser used.
- The performance efficiency expected out of the application for data search and filtration. If the data search and filtration are some of the key drivers of application and database is expected to grow in number of documents and size, a suitable separation of databases should be considered in terms of central and local repositories. For example, the product development team from North America would deal with a database dedicated for that region and likewise a separate database for Europe and other locations.
- A centralized repository of all documents or truncated document in a common database for reporting and escalation purpose at global level. You might, for instance, have similar databases each used in a single country, but automatically copy an extract from each of these into a central reporting database.
- The target audience of the application. For example. if there are few modules to be accessed by third party, then such modules are to be hosted on secured access zone behind the firewall.
- The application expected to be hosted for third party internet users of the organization, For example, the partners and vendors. There are security concerns and legal restriction in terms of data access. A separate database with document available for third party access should be maintained in-line with the organization policy for data access.
- Other access considerations. If your application contains a significant number of documents that should be visible to one group of users but not to others, you can secure the documents with Readers fields, but this both has performance implications, and is complex to implement with certainty that it has been done correctly. By comparison, putting private documents into a separate database and using the database ACL to limit access, is simple to implement and simple to demonstrate as secure.

Module level design

This section describes the features and functionality details of the individual modules. The purpose of this section is to provide the primary functionality of the module. It details the further specifics to any constraints, design considerations, dependencies, interfaces incoming or outgoing, and exceptions to be managed. The data flow in the module in terms of input and output process of the module is described here. The use cases

specific for the individual modules is also defined here.

When working on the module level design for Domino application, consider the following:

- Try to limit the number of views. Too many and too complex views, is the most common factor limiting application performance.
- Try for a balance between the number of forms, and the complexity of each form. It takes a certain amount of experience in Domino development to get a feel for what makes sense to go together in a single document. If two document types are very similar, you can consider combining them. If a single form is huge and contains dozens of hide formulas, optional sections, and hundreds of fields, chances are you're trying to do too much with a single form, and should consider a main document and responses instead.
- Make use of common functions and reusable script libraries to provide the interfaces between the modules.
- Avoid duplicating the data across modules.
  For example, in a product development application, product related information and the related process to maintain it are better managed in one module and the reference view lookup or back-end script should be used to integrate the product data in the other modules. However, there are tradeoffs here. If you have another application store only the product ID when it refers to a product, then whenever users use that form, either they only see the product ID and have to click on something to find out what product it represents, or the application has to do a live lookup with the product ID as key, to display the product name and other information. There's nothing wrong with that, but it has limitations: the looked-up data are not available for use in views, and if there are too many lookups on a single form, performance may be unacceptable. Therefore, you may occasionally need to duplicate data, but consider whether there is a need and how aggressively you have to keep the data up to date.
- Use shared code, sub-forms, shared columns, and shared actions to achieve reusability of similar features across modules. However, do not overdo the use of shared code component because there is a runtime overhead for Domino to link the shared items with other items for final rendering. A few simple rules:
  - Use shared code components if the items could be used at more than one design element or modules.
  - Benefit of maintainability is more compared to the marginal overhead for the Domino to link the shared components.

Workflow process

Notes/Domino doesn't contain any specific feature called "workflow." Workflow is something you do with the product as part of your application design. There are, however, features in the product that are helpful in implementing your own workflow process from scratch. There are also add-on products available to assist you in implementing workflow in Notes, including:

- Lotus Workflow, a product offering from IBM.
- Various third-party products.
- At least one free workflow framework on openntf.org.

The domino application portfolio for an organization may include many applications that are workflow enabled. Building the workflow capabilities in Domino application can take considerable effort depending on the complexity of the workflow. If you do create your own workflow, design reusable components that can save you time in making the workflow functions available on all forms. Ideally you can reuse your workflow components in other Domino applications.

The following features of Domino are useful in building workflow into your applications:

- Reader and Author fields to control access to a document at different stages of its lifecycle.
- Electronic signatures in "signed sections" to prevent forged approvals.
- Scheduled agents to provide email reminders of overdue workflow steps.
- Hide formulas and controlled access sections to limit who can view or edit different parts of a document, depending on its workflow state.
- Document locking, to prevent replication/save conflicts during workflows which include parallelism or where there are multiple possible approvers at a given step.

To design a workflow capability that is easy to maintain with high rate of re-usability using Domino Designer, consider the following:

- Design a workflow engine instead of hard-code all the workflow rules spread across the application design.
  This approach helps the development team in efficient maintenance of the workflow process of the application and its scalability. Making the workflow process of Domino application configurable gives flexibility to reuse it in several applications.
- The workflow engine should have three major components:
  - Workflow configuration documents:
    This is used to deal with the different phases, steps, activities or status of the workflow and associated details. Each workflow configuration entry stores the following details:
    - Each workflow configuration document has configuration key or ID.
    - Module name to which this configuration is associated.
    - Name of the field in document to identify the current actors (to populate Authors) and spectators (to populate Readers) by the workflow engine.
    - Store the next workflow configuration key in case of approval or rejection of the current stage. This attribute help workflow engine define the document routing sequence.
    - The mail subject and body in case of approval or rejection email requirement.
    - The name of field in form/document to identify the names to send workflow emails
    - Several other mechanisms to be considered in workflow configuration, is it parallel or sequential workflow or it’s a mix.
  - Workflow engine script library:
    The library is used to read the workflow configuration parameters defined in workflow master and act on the document as appropriate depend on current stage of the document triggered by users action.
  - Workflow user interface components:
    This are the workflow actions to trigger the workflow step with reference to the respective workflow configuration.

Program specifications

Program specifications should include the details of the GUI design, database design, module design, and workflow process. In the program specification, detail out the different possible Domino design elements required for various purposes that are *input* (Forms), *output* (Views and Forms), *data storage* (Documents), and *Data manipulation* programs (agent, script library, actions, events, and others). The program specification is the instruction for the developer to design and build business logic into the application. The input description includes the navigation, layout, input transaction, data editing rules,

data validations, messages and general input processing. The output description includes the layout, panels, reports, screen output, messages and general processing. The data storage structure deals with the different fields, its data type, the access to data, data calculation or update rules. The data manipulation programs describes about the business logic for creation, update, and deletion of data, workflow processing rules, business logic, dependency of modules, and other components.

Security procedure design

Domino is known for its granular level security control. Every single detail about the application security features to be considered, right from the database access control list to the documents, sections, and fields, the level of different access the business actor has on the database, and access rights of other system on this application are also documented in the program specification section. The following features provide a high level of security against unauthorized viewing and editing of documents:

- Database ACL
- Document Readers fields
- Document Authors fields (for users with Author access to the database - Authors fields are irrelevant for Editors and higher)
- Encryption (to hide individual fields)
- Electronic signatures (which don't prevent editing, but make unauthorized editing easy to detect).
- The $UpdatedBy and $Revisions items of documents, which make it impossible to modify a document without any trace.

There are also several Notes/Domino features which resemble security features in that they hide data from casual users, but are not secure from a "power user," particularly if they have a copy of Domino Designer. It's not wrong to use these features; you just need to be aware that if you need a higher level of security, they aren't sufficient on their own. A few of these features are:

- Hide-when formula
- Restricted category option in view URL, or restrict to category in an embedded view in the Notes client
- Restriction on document viewing or editing using JavaScript onload() redirection in a web application.
- Restriction on document viewing or editing in the Notes client by means of form events, e.g. Queryopen.
- Restricted access to views
- Different type of computed fields
- Other DHTML features of styling the visibility or editing features of fields and sections

As a general rule, if a user technically has access to view or edit a document or field, and they are sufficiently skilled, they can find a way to do so, through writing code that bypasses your UI or often by other, simpler means. If your application demands the highest levels of security, you must use the features in the first list.
To design security procedures for Domino application, consider the following:

- Avoid using the anonymous access to database unless it is required.
- Avoid adding individual user and server names to the database ACL. It makes maintenance of application access difficult. Instead, use groups to simplify the ACL entries.

- There is a limitation of about 75 roles supported through database ACL. Consider using ACL roles for security implementation only if the roles are expected to remain within the limit during the life of the application.
  For example, using role such as country names to grant access to the users of respective country in a global application would lead to a situation of limitation. Use different groups for country or replica copies instead of role implementation.
  Roles are generally best used to control access based on a user's tasks within the application; for instance, there might be a "CostApprover" role with authority to perform a given workflow approval.
- Grant Editor and high level access in database only to the required users.
- Use Author and Reader level access to enforce the document edit and read access to the authorized user of the application. Readers fields have performance impact in views, which is detailed elsewhere in this wiki.

Detailed design document

On the basis of the design elements identified in the program specification, the detail design document must be prepared. This activity is a drill down version of program specification wherein every detail of each field in the form, each column in a view, each action button, hides-when formula, every event, agents, script library, function, every small detail about the application development are mentioned.

2.3.2 Prototyping

Prototyping is the concept wherein subset model of the application attempts to simulate only a very few features of the final application. Prototypes are aimed to get feedback at early stage from the user about the critical aspects of the application. The prototype is also used to generate a proof of concept in case the application feature deals with either new technology or methodology and likewise. Prototyping techniques are used for early validation of operational concepts and user interface. It is also used when the user are not sure about how to implement a certain set of features. It also helps to validate the feasibility of a solution. Domino Designer, being a Rapid Application Development tool, is fast and facilitates rapid iteration between prototypes.

2.3.3 Test planning

Test planning identifies the business requirements for which the system is developed and also specifies the application development aspect. It identifies activities, dependencies, and effort required to achieve the test objectives. The test planning adopted is based on the risks involved in order to make optimal utilization of test efforts. A manual testing of Domino application is guided through the test case document prepared. The test cases are the business scenarios for which the developed application must be tested against.

The test plan describes how the testing will be implemented. It should state what will be tested, the major actions taken while testing, test techniques that will be used, the extent or depth of the test, and how the results will be verified. This approach shall ensure that the requirements are adequately tested.

The criteria for testing has to be well defined. It states the criteria for accepting and rejecting the test results. In case of unit testing, the code review should be completed and the associated defects detected in the code review cycle should be fixed. The entry criteria for integration testing should be the unit tested code, with the defects detected in

the unit testing phase fixed. In case of systems testing, the code should have completed the Integration testing cycle, with the defects detected in the Integration testing phase fixed.

The application module tested may require another testing cycle if the variance between the planned and the actual number of defects detected in the testing process is considerable or a major defect is encountered during the testing process.

It is crucial to plan to test the application with a realistic number of sample documents. It's very common for Notes/Domino applications to work fine in testing when using one workstation with a few dozen documents, but when they're in actual use with tens of thousands of documents and hundreds of users per server, performance bogs down. It's also a good idea to have a realistic number of (automated) test users for performance verification.

2.3.4 Design authorization

The design authorization is an important step to obtain a buy-in from all the stake holders. The initial design authorization is done within the hierarchy of the development team. This final internal review of the design is done before it is presented for customer sign-off.

2.4. Development

In this section we have a look at how you can fine-tune your coding and development process to optimize your application development and its performance.

2.4.1 Coding practices

Best coding practices for software development can be broken into many levels based on the coding language, the platform, the target environment, and so on. Using best practices for a given situation (in our case Domino and XPages) reduces the probability of introducing errors into your applications, regardless of which software development lifecycle (SDLC) model is being used to create that application. In a team environment, best coding practices ensure the use of standards and uniform coding, reducing oversight errors and the time spent in code review.

Some of the best coding practices that you can use are:

- Use a uniform standard and naming convention across all the code.
- Have code reviews involving multiple developers.
- Unit test all code.
- Have proper documentation, including generous amounts of code comments.
- Use script libraries to prevent duplication of functions, constants such as view and item names, and common datatypes and enumerations.
- If you write LotusScript or JavaScript, use object-oriented coding techniques.
- Avoid goto statements and overly deep levels of nesting.
- Limit modules to a reasonable length; generally not more than 60 lines.
- Simplify and optimize your code as much as possible.
- Remove unnecessary code before deploying.

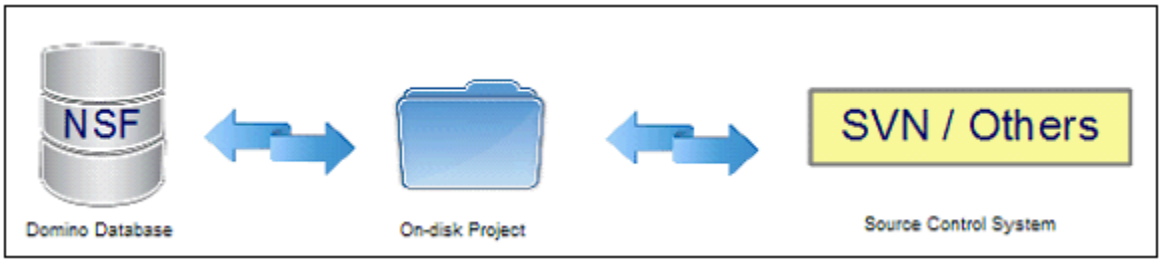
- If you write LotusScript, use Option Declare, which cuts your initial coding errors by half. There's an option in the editor which automatically inserts this statement for you.

For general coding best practices, we recommend doing some reading on the subject, for instance in such works as The Elements of Programming Style by Brian W. Kernighan and P. J. Plauger.

We discuss more about XPages development best practices in 4.6 Optimizing XPages using custom control.

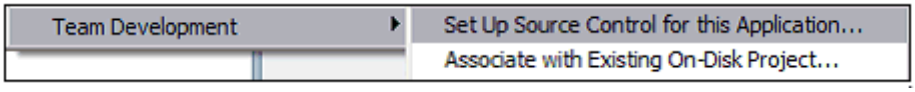
2.4.2 Version management

For an effective tracking and proper maintenance of your code, you should use a version management system for your applications. In Lotus Domino Designer 8.5.3, a new source control feature has been provided which helps you to do version management out of the box without using the external plugin that you had to use earlier. This feature has been introduced to let you store NSF based design elements into a source control repository and perform common source control operations using Lotus Domino Designer. Using this feature, you can create an association between an on-disk project managed by the source control and the virtual NSF based project that your Designer recognizes and tracks.

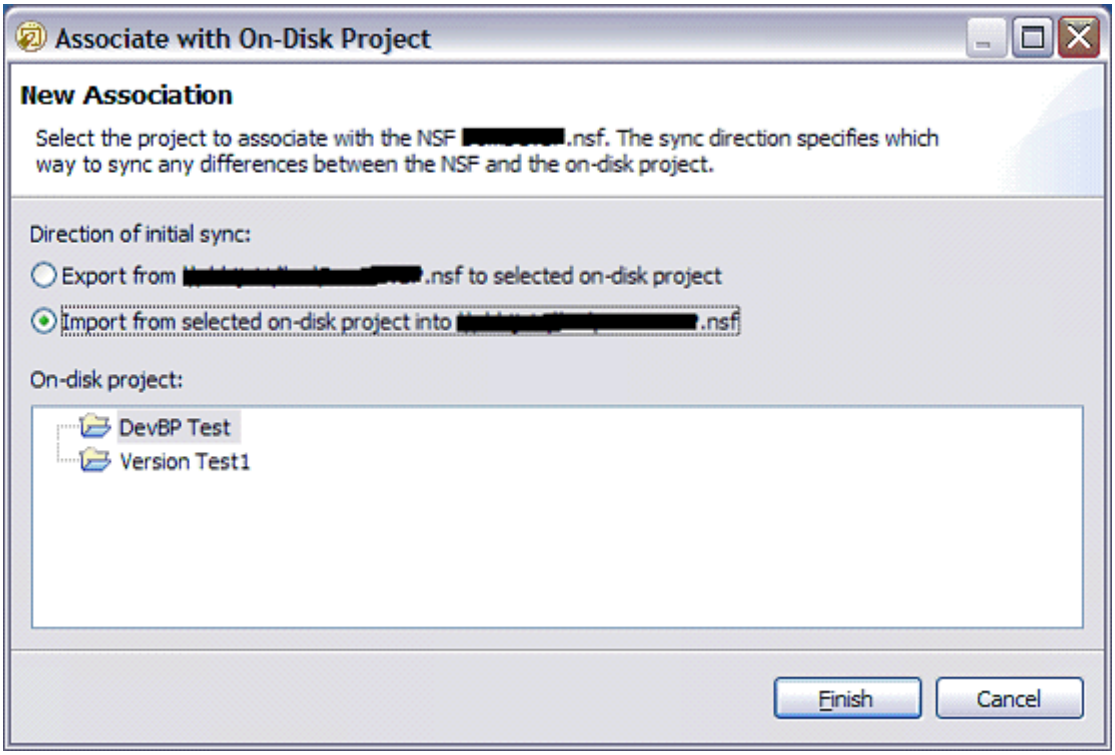


To use this feature, follow these steps:

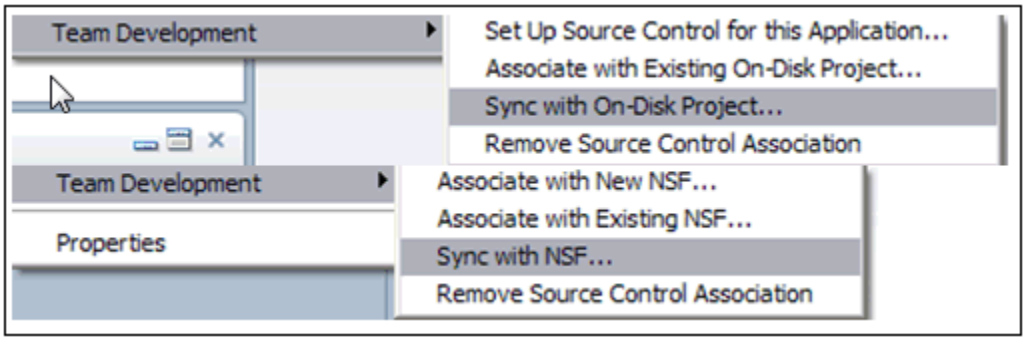
1. Install and configure the open-source version source control systems and create and configure a repository.
2. Associate your application (NSF) with an on-disk project (ODP) or create a new one.



3. Commit your changes from Designer on-disk project to the repository.
4. Ask your team to retrieve on-disk project from the repository and associate their databases with the on-disk project.


Associate with On-Disk Project
New Association
Select the project to associate with the NSF [redacted].nsf. The sync direction specifies which way to sync any differences between the NSF and the on-disk project.
Direction of initial sync:
Export from [redacted].nsf to selected on-disk project
Import from selected on-disk project into [redacted].nsf
On-disk project:
DevBP Test
Version Test1
Finish
Cancel

5. Modify either NSF or on-disk project (ODP) and synchronize the two files.



6. Commit changes to repository and ask other team members to download and sync their local files.

2.4.3 Unit testing

Unit test is the first and most important level of testing in the software development lifecycle. It is usually done by the programmer, and it is their responsibility to make sure that their piece of code passes through without any errors and is compatible with the rest of the application. You can save a lot of effort and resources in the long run if all the programmers in your team do their quota of unit testing thoroughly.

In order to make the unit testing effective, first of all, make sure that it is followed rigorously by all the team members and have and allot sufficient time for them to test.

You can refer to this link for details on what, why and how of unit testing

http://en.wikipedia.org/wiki/Unit_testing.

The Domino open source website, OpenNTF.org, contains multiple projects with free tools to assist in unit testing of Notes and XPages applications.

2.5. Test

The test phase ensures that the planned tests are executed to ensure the quality of the application developed. It includes setting up the test environment and test data. The test setup should simulate the real environment of the business as close as possible. The scope of testing would be limited to the test planning and test cases prepared. The test environment should have all the dependant system and interfaces configured for proper testing. The testing phase is bound to be conducted in an iterative form where any defect observed during testing must be fixed and re-tested again.

The Domino application test setup includes several aspects:

- Setting up the test server configured in line with the production and pre-production servers for better simulation of test scenarios.
- Creating multiple test user IDs on server to test application for different set of users with varied access rights.
- Setting up of an appropriate database access control list (ACL) as per the design specifications to ensure each test user ID has an appropriate access rights. Avoid using the higher access in database ACL for any test user ID that does not match the user role defined in the design specification.
- Deployment considerations: If the Domino application is to be ported to several platforms, for example, multiple version and types of browser, mobile and smart devices, an appropriate test setup should consider all these factors and the test environments should be made available for testing.
- Internationalization considerations: If the Domino application is to be deployed at a global level, providing multiple servers with different time zone and regional settings is another key factor of setting up the test environment to simulate the real life environment.

2.5.1 Application testing

This phase consists of unit, integration, and system tests. All the modules are tested individually and then integrated for the system testing in successive cycles. The defects found in a cycle of testing are fixed before commencing the next cycle of testing. The systems testing is also carried out as part of this phase. There should be no open issue which is critical for the user acceptance the application. The application is not handed over for user acceptance test (UAT). Unless all the issues reported during the application testing are not fixed, verified, and closed.

The key aspects of the applications test are:

- Conformance to business requirements:
  This aspect of application test is focused towards to ensure the application developed meets all the business requirements. The requirement in terms of GUI, business use cases, business rules, and all type of business related processing expected from final application. Verify the features and the functioning of them as per the requirements and design specifications.

- Process workflow and application security:

The important aspect of most of the domino applications is its workflow and security implementation. This being a core of any domino application, a special attention is required in this area including the workflow routing of the document and mail routing associated with it and the security implementation in terms of authors, readers, controlled section, and data security at different levels of workflow. The workflow must be tested with test user IDs having appropriate access to simulate the business environment.

- Application performance test
  Domino applications are expected to be tested for the performance. It encompasses several aspects of performance testing of an application: the response time, screen loading, form submission, the processing efficiency of various features, and overall performance with peak load of concurrent user using the application.

- Multi-browser compatibility test
  Consider the browser standards of the target business organization for Domino applications. In genera, business offers the application compatibility to one or few of the popular browsers. The application has to be tested for compatibility and awareness of the types of browser that offered. Perform the proper test of each browser enabled feature on the target browsers to ensure the application compatibility for the same.

- Multi-language or regional setting test
  Domino applications must be tested as per the various target user's regional settings to ensure better adaptability. The scenario wherein the application hosted on a Domino server that has a different regional setting as that of the users using the application should be established. A specific consideration must be made to test the date, time, and number type data fields for the formats, calculation, and presentation. The test is also required to understand the different character set to be managed for any data input and output processes for the application.

The Domino application testing is one of the key phases in the Domino application development life-cycle. The following are some of the guidelines to test the Domino application.

GUI
The GUI of the Domino application should be tested for its consistency in screen layout, font formatting, and color scheme as per the design specifications or the GUI standards defined during the design phase. Test the GUI screen of a view as well as of forms with a possible data in the Lotus Notes document with inline HTML or mark-up text entered by the users in one of the field. The system should either render such inline mark-up text without disturbing the application GUI or validate such data entry with appropriate restrictions.

Data caching
Test the Domino application for a consistent, recent data presentation on screen for dynamic contents. Post a Lotus document with data content from browser to Domino server and re-open this document in read mode in browser. Verify that the last modified data, not the cached data, is display on the screen for the document. Irrespective of the browser setting for cache, the Domino application screens that deals with the dynamic data should be tested for no cache data implementation.

Data views
Test all the user interface views of the Domino application for the number and the sequence of columns required as per the design specification. You should also test the

views for other aspects such as view navigation or pagination, column sorting, default view sorting, document listing, empty categories (in case of categorized views), view actions, and view filtration. Test the view data access, log in to the Domino Application using different user access rights or user IDs and test for restricted document level access. Test the response time of the navigation across pages for the views to make sure it meets the requirement. The view level actions should be tested based on the features enabled for the Domino application. Such as the in-view document edit option, processing of the selected documents, and any other specific view actions must be appropriately tested with reference to the design specifications. The new documents created in the database which qualify to be listed in a particular view should be tested for appropriate listing of the newly created document in the view to verify the view index is automatically refreshed. If the document contents are expected to have multi-lingual data which includes accented characters, test the view column sorting for accent sensitive sorting.

Data access, edit, and search
Test the data manipulation features of the Domino applications for data validation, data consistency, data constraints, and business rules expected as defined in the design phase. The data accessible through a form in different sections as well as the fields and actions expected to be visible or actionable to the users with different access levels must be tested as per business rules defined for the application. For thorough testing, prepare an access control matrix for each type or level of access rights and roles against each and every user interface action and data presentation layer. Prepare such matrix in line with the design specifications and the business rules defined during the design phase. This approach would ensure that data access, in terms of visibility and editing, is tested appropriately without missing any particular combination of access levels expected from the application.

The data search capabilities provided in the Domino applications must be tested on the following parameters: the data fetch response time, the data search accuracy, and the data security. During the test environment preparation, generate the kind and the amount of test data as it is expected in the production environment. Work with the business users to obtain this data to simulate the proper data search test scenarios. Avoiding this test could result in a number of blocking issues during the user acceptance test (UAT). Generally, in the UAT phase, the business user would use the actual data from the business to validate the user acceptance of the application. Hence, at this stage of application testing, it is critical to not only test the application features but also use the business data to simulate the actual business environment. This approach would help identify applications acceptability in terms of handling different type and set of data content expected to be used during production run of the application, for example, the amount of data to be managed by the application considering the initial production run and the data size growth expected during the applications production life-cycle.

The UAT incudes testing the applications response time and efficiency in data search capabilities of the application. Verify the performance requirements defined by the business or the design team in design phase is met with the data search response time parameters in the performance benchmark. In addition to the response time, the data search accuracy test is equally important in UAT. If the full text search feature is implemented to gain performance efficiency on data search response time, it has its own set of inherent flexibility which might not be acceptable from data search accuracy point of view. For example, the search action triggered (using full text search) for searching text "Belgium" would fetch the document field contains "Belgium" as well as "Kingdom of Belgium". This might not be the result the business users expect. It is important to understand the expectation of the business for such scenario, wherein whether the business needs the exact match or word match.

Background agents
Domino applications implement Lotus script and Java agents that have to be tested to verify the expected functionality defined in design specification. Test the background agents for the appropriate access rights for the business requirements. Generally, a background agent is run using the access rights as that of the agent signer. However, in many Domino applications, it is required that the agents to be run as web user to allow the agents to execute with access rights of the current session user. For example, the WebQuerySave agent that triggers email or performs certain activities that require the current session user context are configured to run as web user. If this perspective is missed during the application testing, it leads to failure in certain business use cases during UAT. This is one of the important aspects to ensure the agent is tested appropriately either in context of agent signer or the current session user as per design specifications. Similarly, depend on the application requirement, the proper functioning of the restricted actions by the background agent must be tested, for example, the proper functioning of the background agents that handle file system operations.

Other business scenario for background agents is where an agent attempts to access database on another server in the same domain. The common Domino server administration policy is to disable the access of any background agent triggered from the Domino server to access resources on another Domino server where the two servers are not explicitly trusted even in the same domain. The solution that considers the background agent accessing resources across servers should be done at earliest during the application development lifecycle. The design phase is an appropriate stage to make such consideration. The Domino application test has to consider this aspect and to check any possible issues arising due to the infrastructure setup constraints or the solution implemented. The best approach is to check the possibility with the administration team if, within the organization policy, the trusting of server in same domain is possible to allow this solution to function properly. The alternate solution is to create a replica of the other database on the source server to ensure the dependant database is available on the same server.

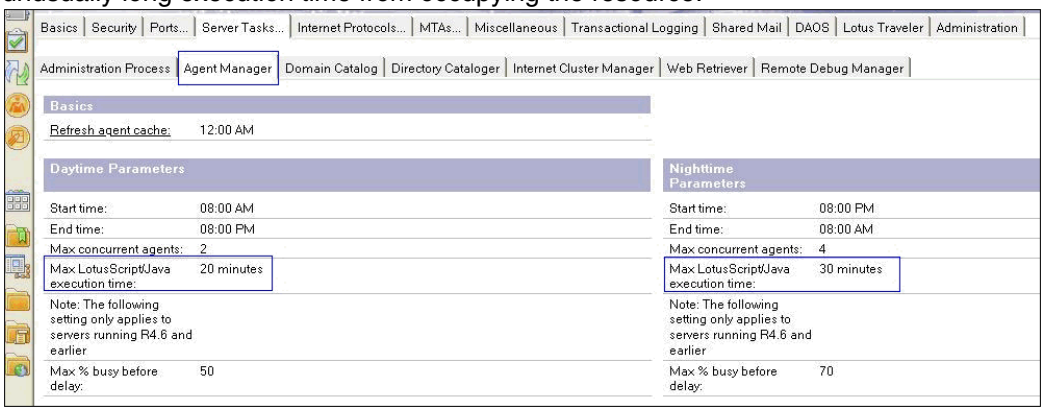
The background agent performance must be tested within the context of the real business scenario. Certain Domino server configurations settings have an impact on the time allocated for any background agent process of Domino application to complete its run on the respective Domino server. The Domino application background agents must be tested for its performance efficiency to comply with the configuration settings of the target production server.

Following are some of the key server configuration settings for a background process of Domino application:

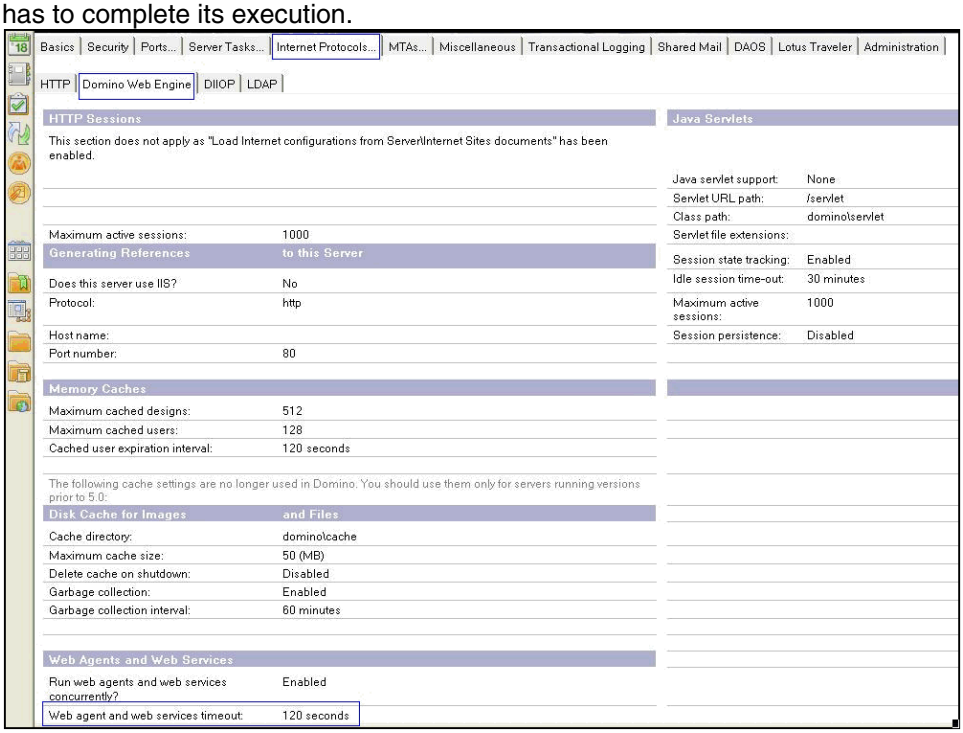
- Max LotusScript/Java execution time:
    - This configuration parameter (see the figure below) in Agent Manager tab has two possible values in context of daytime and nighttime parameters. The agents run during each period are subject to the settings in the parameter fields for that period.
    - Agent Manager is a server task that is responsible to manage the schedule and background agents run on server.
    - This parameter specifies the maximum minutes a Lotus Script or Java agent has to complete execution during that part of the day. If the agent exceeds this maximum execution time, the job won't be finished, and the Agent Log records the termination.
    - Defaults are 10 minutes for daytime and 15 minutes for nighttime. Domino administrator can increase the execution time limits to allow the complex agents sufficient time to complete. Setting the value of this field excessively

high might impact system performance, because the Agent Manager runs for a longer time to accommodate the agent.

- This time limit setting prevents erroneous agent processing that take unusually long execution time from occupying the resource.


Basics | Security | Ports... | Server Tasks... | Internet Protocols... | MTAs... | Miscellaneous | Transactional Logging | Shared Mail | DAOS | Lotus Traveler | Administration
Administration Process | Agent Manager | Domain Catalog | Directory Cataloger | Internet Cluster Manager | Web Retriever | Remote Debug Manager
Basics
Refresh agent cache: 12:00 AM
Daytime Parameters
Start time: 08:00 AM
End time: 08:00 PM
Max concurrent agents: 2
Max LotusScript/Java execution time: 20 minutes
Note: The following setting only applies to servers running R4.6 and earlier
Max % busy before delay: 50
Nighttime Parameters
Start time: 08:00 PM
End time: 08:00 AM
Max concurrent agents: 4
Max LotusScript/Java execution time: 30 minutes
Note: The following setting only applies to servers running R4.6 and earlier
Max % busy before delay: 70

- Web agent and web services timeout:
  - The web agent or web services in context of this parameter (see the figure below) is the agent and web service that is triggered by the browser clients. This include application agents invoked by the WebQueryOpen and WebQuerySave form events and for the agents invoked by the URL command **OpenAgent**.
  - This setting allows the administrator to manage the execution time limit for web application agents. The purpose of the time limit is to prevent web agents from running indefinitely and using server resources. This parameter specifies how the maximum number of seconds a web agent or web services has to complete its execution.


Basics | Security | Ports... | Server Tasks... | Internet Protocols... | MTAs... | Miscellaneous | Transactional Logging | Shared Mail | DAOS | Lotus Traveler | Administration
HTTP | Domino Web Engine | DIIOP | LDAP
HTTP Sessions
This section does not apply as "Load Internet configurations from Server\Internet Sites documents" has been enabled.
Maximum active sessions: 1000
Generating References to this Server
Does this server use IIS? No
Protocol: http
Host name:
Port number: 80
Memory Caches
Maximum cached designs: 512
Maximum cached users: 128
Cached user expiration interval: 120 seconds
The following cache settings are no longer used in Domino. You should use them only for servers running versions prior to 5.0:
Disk Cache for Images and Files
Cache directory: domino\cache
Maximum cache size: 50 (MB)
Delete cache on shutdown: Disabled
Garbage collection: Enabled
Garbage collection interval: 60 minutes
Web Agents and Web Services
Run web agents and web services concurrently? Enabled
Web agent and web services timeout: 120 seconds
Java Servlets
Java servlet support: None
Servlet URL path: /servlet
Class path: domino\servlet
Servlet file extensions:
Session state tracking: Enabled
Idle session time-out: 30 minutes
Maximum active sessions: 1000
Session persistence: Disabled

2.5.2 User acceptance testing

This phase consists of acceptance testing by the customer. Development team must fix defects found during this period by the users. In addition, the development team has to continue to test the product and improve quality. The acceptance test has to be performed for all deliveries and releases.

2.6. Deployment

The Domino application deployment procedure varies among the organizations. In this section, we discuss the some of the best practices pertaining to the Domino application deployment in an enterprise. Application deployment is planned and scheduled as part of the project planning. The stakeholders (business owner of the application, application development team, and Domino administration team) make the decision of deployment date and action required for successful deployment. Decisions include the planned downtime if required, packaging template (.NTF) for moving to production, and a detail activity plan. The Domino application is eligible for deployment only after the positive sign-off received for the application in user acceptance testing (UAT) phase. In the majority cases, the responsibility of application deployment to production is with the Lotus Domino Administration deployment team.

Deploying new release
When deploy a new release of an existing application on the Domino production server, consider the following tasks:

- The application level downtime is required.
    - Application business owner plans for the application level downtime that is suitable for all the stakeholders, For example, the deployment could be schedule on non-business working days or time to have lesser impact (it is the best for a complex deployment which runs into several hours).
    - Inform the impacted users through email about the application downtime at least a week in advance to allow the business users to plan their activities accordingly. It is a good practice to send a reminder a day before the scheduled deployment.
- System administration team plans for proper backup of production copy of the impacted Domino application databases with data.
    - The deployment should prepare a rollback plan (if required consult with the development team for inputs)
- Domino administrator identifies the template for deployment from the UAT environment that have received positive sign-off by the user.
    - This template is to be used for design refresh or replace on the production database as part of the deployment activity.
- Development team prepares deployment related activities including required data updates on the existing documents, configuration creation , required batch processing of documents, deployment verification, and so on.
- On the day of deployment, Domino administrator perform the following activities:
    - Revoke the access right of the general users from the impacted database[s].
    - If the domino application has several replicas across servers, disable the replication temporarily on the hub replica for deployment activities. It helps to avoid any replication conflicts and database inconsistency while deployment process is initiated.

    - Refresh or replace the production database with the final template (.NTF).
    - Check for any errors with reference to the design refresh or replace action.
    - Update the view indexes and recompile all the scripts.
    - Perform any change in ACL if required, for example, add, edit, or remove roles, ACL entries, and ACL privileges.
- Development team performs deployment actions on the application databases. For example, perform batch update of existing document (add or remove of fields, or change in key field value), create or update application configuration.
- Development team verifies the deployment by making observation of key features and screen as expected.
- Have the user representation for the critical application deployment to ensure timely identification of any issues before the application is made available for general usage.
- Upon the successful completion, perform the following actions:
    - Domino administrator re-opens replication of the application on hub server.
    - Domino administrator restores database access to both existing and new users.
    - Application business owner sends an acknowledgement email to the impacted users about the successful deployment.
- In event of failure of the deployment activity, Domino administrator invokes the rollback procedure to restore the backup on production. Share the failure report to the stakeholders.

Deploying a new application

For a new application, the application level downtime is not be required. The deployment tasks include:

- Creating the NSF files for the application from the template file from UAT.
- Setting up the access control lists for the NSF files.
- Replicating the NSF files to a Domino server, or e-mail them to somebody who has the permission required to do so.
- Setting up any data in documents that the application code required to function properly.
- Application business owner sends an acknowledgement email about the launch of new application.

Deploying a minor fix

When deploying a minor fix, the application level downtime might not be required. The deployment process is simple:

- Consider the criticality and complexity of the application and back up the databases as the rollback plan. You can use the daily regular backup copy if available.
- Domino administrator simply refreshes the design of the impacted databases.

2.7. Production

In today's Domino environments, the number and the complexity of Domino applications are increasing due to growing demands and due to the rapid and easy-to-use application development technologies provided by the Domino Designer that facilitate more developers to build applications and to deploy them to an existing infrastructure. In an IBM Domino environment with a large number of applications, one of the main question is how to support this environment so that all involved parties can work efficiently. In this section, through explaining an application hosting framework, we guide developers, administrators, and database owners for improving services so that an environment can scale from a small to a very large deployment while increasing functionality, the level of automation, and customer satisfaction. We also provide the Domino clients guidance to avoid problems before they occur.

Our discussion about Domino application development life-cycle in the previous sections covers the following items

- Quality control for applications that are being developed or being extended.
- Standards or design guidelines for Domino applications.
- Interaction and communication between application owner, application developer, and administrators of the environment.

For production environments, including Domino environments, issues might be raised include:

- Resources required by existing applications are growing over time due to more users or more data within them.
- Operational costs of an application are not identifiable
- Application life-cycle is not supported, not controlled or verified, and not automated.

Due to these issues, a number of effects might be the result for the operation of an environment, which for example, has an impact on user satisfaction or the total cost of ownership of the environment:

- Unable to plan infrastructure resources such as CPU, storage, and memory consumption up front.
- Wasting resources on inefficient applications while others would need them.
- A large variety of custom developed applications due to missing implementation or release processes.
- Critical applications did not get the resources needed.
- Troubleshooting and ad hoc actions become more and more a priority for operators to face low performing applications.
- Simple and efficient applications have to share the costs with the heavy and complex applications that did not use best practices at design time.

Based on the as-is situation described above, the goal of this framework is to significantly improve user efficiency with the following items:

- Improving serviceability by clearly separating interfaces, processes, and services

- Having clear guidelines for all participating parties within the life cycle of an application by using properly defined standards for server infrastructure and applications
- Avoiding application issues (which is equivalent to improve reliability) by proactively planning resources and by continuously reviewing existing applications
- More transparent billing process by classification of applications into categories

While the technical aspects of operating Domino servers has been covered already in the IBM Redbooks wiki Optimizing Domino Administration, this document focus on the application management process in a production Domino environment:

- 2.7.1 Application management strategy
- 2.7.2 Application management process

2.7.1 Application management strategy

An application management strategy describes an high level approach about how to handle Domino applications within an organization. It is more a concept that can be extended from small implementations to large scale enterprises rather than a fixed set of settings and properties. The application management strategy items described in here are supposed to define the steady state operation for a Lotus Domino environment with a clear goal to improve service quality and process efficiency.

Let's start by clarifying the different services and standards which apply to a production environment.

Infrastructure services

Infrastructure services describe the components which are required to operate a Domino server regardless of its applications. Those services are to be in place before one can start putting production applications on a server.

- Installation of a Domino servers and additional resources:
  If required, the infrastructure team is requested to install additional servers according to predefined standards (CPU, RAM, HDDs, and so on).
  Defining server categories (small/medium/large) and assigning a specific hardware configuration for each type is helpful here.
- Monitoring of Domino servers:
  Continuous monitoring of running server tasks and server load. Monitoring includes taking care of CPU, memory, and disk utilization.
- Service level reporting:
  Reporting of service levels on a per server basis. If needed, those can be reported for each class of applications.
- Backup and recovery:
  Continuous backup of application data with defined retention periods according to a defined backup plan. Recovery of complete servers or individual applications after a crash or data loss according to defined procedures.
- Virus scanning
  Regular scanning of application data for viruses including definition about how to handle applications which contain harmful code.
- Troubleshooting
  In case of a problem, identify the root cause and take actions to return to steady state

operation as soon as possible.
Analyze the server outages and error messages generated from server crashes.

In order to establish these infrastructure services standards and service levels, a clear description of each element shall be agreed between the involved parties.

Application services

Application services describe the most important services required for operating applications on a Domino server. Those services are typically provided by an application development or application support team in close cooperation with the administration team.

- Maintain an application repository:
  Such a repository is a list of Domino applications with its metadata such as application owner, file name, replicaID, ACL information, operational requirements, expiration date.
  This repository should also contain a classification of the application. Key element is to assign an owner for each application, which can be contacted at a later stage, for example, in case of questions about access rights to the application.
- Rollout of applications:
  Describes the implementation of new Domino applications or additional instances of existing applications according to predefined processes.
  These processes include the collection of metadata in the application repository as well as categorizing applications, quality control, and implementing them into monitoring systems and backup.
- Operation of applications:
  To operate an application, there is more than just providing CPU, memory,or disk capacity. It means operational tasks that are related to the application itself, such as full-text indexing, scheduling agents, or taking care of add-on tasks which have been installed on the server to support this application.
- Troubleshooting:
  Analyze error messages and warnings in case of outages or application errors that lead to partial or complete unavailability of the application.
- Periodic review:
  Regular (for example, every three months) verification of application parameters, data growth, number of users accessing an application and its performance impact. This includes to check if an application that originally was developed as a small-and-simple application requires to be optimized due to its growing number of users.
- Application change management:
  Implementation of changes in production applications, version updates, or functional enhancements should be performed by applying change management rules.
- Access management:
  Change access rights to an application by adding members or roles to the access control list (ACL).
  Note: Maintaining access rights is to be done by the owner of that application who should be allowed to modify the group members, but not be allowed to modify the ACL.
- Creation of usage and billing reports:
  Supply information about the usage of an application to the application owner on a regular basis.
- Rollback and archive:
  On request of the application owner, delete or archive applications from a system.

Not in scope of the application services is the training of users, or the development of individual applications.

Application standards

Application standards in this context lay the foundation for a streamlined operation. By defining and writing down these standards, the operation team and developers will have a common understanding of what rules do apply to the Domino applications in production. Those rules can describe a variety of elements that apply to a production environment. Here are a few examples of what application standards you can define:

- Organizational:
  These type of standards define, for example, who owns an application, how to handle situations where the owner is changing, what is the procedure for creating additional replicas on other servers, and what is to be done for removing an application from the server.
  Do not think about the technical actions, think more about the communication and process point of view.
  One highly recommended organizational standard is to define that each application must have one application owner. Also describe what happens if this is not the case, for example, "if the owner is not identified, the application will be removed from the system by the end of the month."
- Access rights
  What should an ACL look like, are individual user names allowed, or do you only use groups. If so, what naming standard should be applied?
  Is it allowed to create personal or shared views at runtime? What is the maximum access right for typical users, here you do not want to have anyone with Manager or Designer rights.
  What is the maximum access right for developers and application owners in production? Again, in a production environment, only administrators should have design authority.
- Properties**:**
  Take a look at the properties of a typical Domino application, and define which of those properties are the "standard" in your environment.
  Describe what a setting should look like and why you have decided to enable or disable it. Discuss individual settings with your peers. For example, What should a full text index look like by default? Do you want to index attachments by default? Want to index encrypted fields? What update frequency do you want to use?
  The developers and administrators might have a different understanding because additional functionality often goes along with an increased server impact in one or the other way.
- Design:
  Define a set of common rules that apply to the design of an application. This is not meant to be a complete coding standard definition, but should define corporate design elements, preferred user interface elements, colors, and themes. One example could be to use the IBM OneUI theme for all newly developed applications.

Standards for Domino servers

Similar to the standards for applications, the standards for Domino servers describe how a servers are to be configured in production. Focus is not on exceptions, its more about the settings that will be applied to new servers once they are set up. How to document standards for Domino servers is explained in Chapter 11 1.1 Documentation of the IBM Redbooks wiki "Optimizing Domino Administration".

Application classes

As part of the application hosting strategy, Domino applications should be separated into classes to provide optimal resources by applying different hosting strategies. Categorizing an application into a class is an important factor for optimizing the infrastructure and reducing total cost of ownership (TCO). For that reason, the class of an application cannot be determined by looking at file size, number of documents, or number of users alone.

Small applications that have a small number of documents might be small in size but have a bad application design that can lead to higher resource required than a well-designed large scale Domino applications with thousands of users. By defining classes, application developers and infrastructure teams can work together to optimize the Domino hosting environment for best performance and user satisfaction. For example, a large number of applications that have a low infrastructure impact can be placed on the same server, while heavy applications with special needs can be given the optimal resources by placing them on the servers with just a few other applications or even hosting them on the dedicated servers.

The following categories are examples that can be extended or shortened according to your needs:

- Standard:
  Are most likely based on a (corporate) template and have the following properties:
  - No or only small number scheduled background agents.
  - Number of views <50 and low frequency of document changes, this being an indication for the view index impact.
  - No interface to third party systems, external communication, DLLs, or this sort of special items.
  - Fully supports operational standards for backup, monitoring, and support, and is inline with infrastructure standards (for example, Domino version, operating system, and son on)
- Advanced:
  All of the previous class, plus one or more of the following items:
  - Are not based on a standard template.
  - High amount of users who frequently change documents, this being an indication for the view index impact.
  - Unable to work in a cluster.
- Complex
  All of the above plus one or more of the following items:
  - Large number of views that are sorted in multiple ways.
  - Require add-on programs such as third party libraries that are not part of your standard Domino server.
  - Do not comply with the hosting model as they require special handling, for exmaple, a backup concept, or custom replication path.

Once again, the list above is an example, you might want to customize it to your needs. Most important is to describe how to identify which category an application belongs to so that everyone involved in operating an application will have an understanding of these categories. Over time, these categories can be further enhanced so that it is possible to automatically scan an application to identify its category.

Billing framework

Depending on your organization type, you might not need a billing framework because all costs are just covered by your organization. However, certain, especially large-organizations, want to recover the hosting costs from those individuals who are using it, meaning they would like to charge back the cost for each application. Because the size of a database does not indicate how much CPU power it will consume, a key problem here is to find a fair method of identifying the costs of the application. Such a billing framework should therefore, defined on corporate level with measurable criteria that allows all stakeholders to take action for reducing costs.

Describe which building blocks apply to your organization, and if, where, and how you will charge for the Domino applications. This clarification should separate one time costs and recurring costs.

One time costs might apply for:

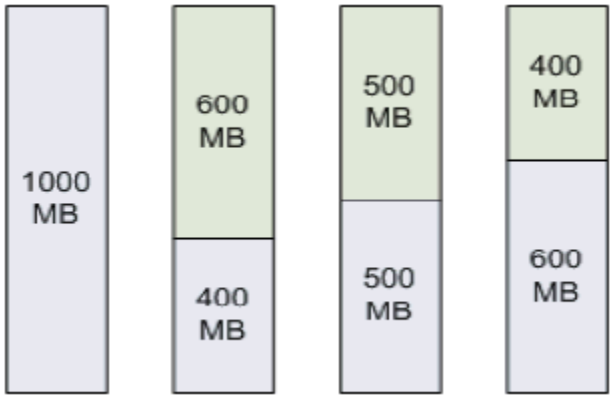
- Rolling out a new application in a production environment
- Version updates
- Archiving or rolling back an application

The recurring costs might apply for :

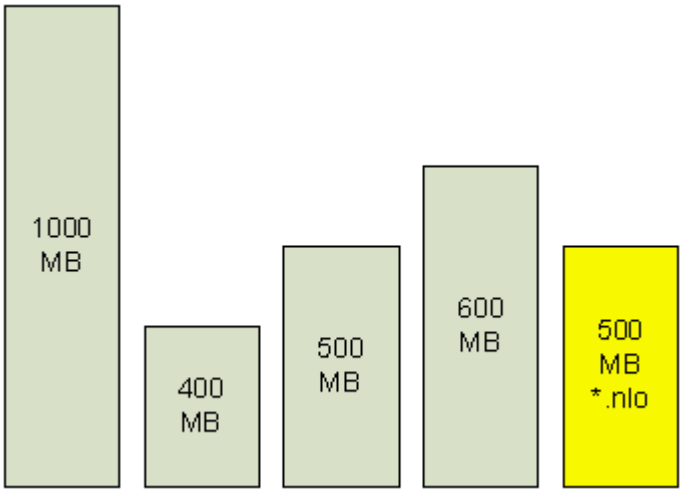
- Infrastructure cost, for example, server and network usage, backup, and monitoring
- High availability (clustering)
- Storage consumption
- Troubleshooting and support

An approach to define a billing framework is often done by using the application file size as an indication for its real cost. Although developers and administrators know that this is not the perfect criteria, its often done this way for simplification of the billing process. By using Domino Attachment and Object Service (DAOS) in Domino 8.5, this might become an issue because the real size of the NSF file does not indicate its logical size, nor the logical size indicates how much storage is really used by this application on the servers disk.

This example shows four applications, each 1000 MB in size, but each with a different number of attachments. While the first one does not contain any attachment, the second contains 600 MB attachments, the third 500 MB, and the fourth 400 MB. For each the size displayed to end users is 1 Gbyte.



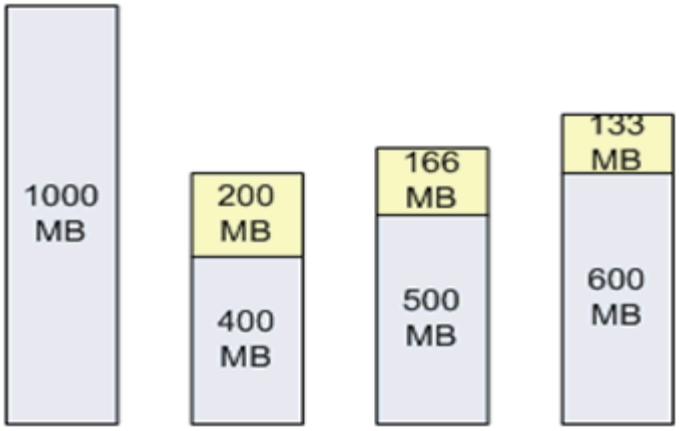
Where in reality, attachments are not stored within the NSF file in all cases. In our example, the first application is not enabled for DAOS and all other are enabled to participate in single instancing. So each of those applications is using a different share from the DAOS storage. Therefore, charging four times 1 Gbyte would not reflect the reality.



The following approach can be a resolution to this problem:
Lets calculate storage consumption based on the real savings from using DAOS. Savings from single instancing are calculated per server and result in a "DAOS ratio".
This DAOS ratio is calculated by taking the total size of all DAOS objects dividied by (Logical size of all files – Physical size of all files).

For each application, the individual storage consumption is calculated based on the following formula:
physical file size + (Logical Size – Physical Size) multiplied by the DAOS ratio which was calculated above.

As a result, each application will be reported with a file size that reflects the storage used on the server.



The side effects of such a storage reporting system are:

- Owners are not able to verify storage consumption or applications themselves because they cant see the total amount of DAOS storage used by that server.
- Owners can only see the logical file size that is more than he is really using.
- Quota calculation (which may apply to applications) is done based on the logical size.
- The DAOS effect will differ from server to server. If the same application hosted on different servers, it will (most likely) not have the same DAOS savings, and therefore, showing different numbers in a report.
- Large servers with a lot of applications will most likely have a bigger benefit from DAOS than small servers.

Conclusion

When all these standards and procedures are described and are published so that everyone in the organization can access them at any time, a foundation for smooth Domino application operation is set. Describing the background and process details will help to streamline application operation.

2.7.2 Application management process

While the application management strategy in the previous section focuses on the infrastructure services that lay the foundation for applications, in this section, we provide the information about application specific processes in a production environment.

From QA to production

When an application has been *fully* tested and approved to be ready for production deployment, the developer or application owner will initiate the rollout process.
An application is handed over from the quality assurance (QA) environment to production by using one (or more) templates and a set of instructions for the administrator to apply the template to an existing application or to create a new application based on this template. For more details, refer to: 3.1 Using development, QA, and production servers.

Since this process is most likely the same every time a new application is rolled out, you can automate this process by building a custom application that helps to manage this workflow.

Furthermore, there are application management tools for Domino available on the market that help to simplify and streamline this process.

When deploying an application from QA environment to the production system, the administrators should consider the following:

- Register each application in an application repository, which describes the purpose, ownership, location, and interfaces of this application.
  Not that this is not the Database Catelog (catalog.nsf) that Domino servers maintain automatically.
- Stay in contact with application developers, work together to improve efficiency of the environment and resolve issues in a cooperative way.
- Store the templates and instructions supplied in a separate application.
  This could be the application repository or another different application and is useful to review code changes that have been applied over time. Such a template repository will keep track of versions and might be shared among the developers and administrators.
- Use unique file names for applications, this will help to avoid problem in a multi-server environment where applications replicate and look up each other, for example, by referring to the file name.
- Use groups for access control list (ACL) management, and allow application owners to maintain those groups.
- Use a special signing ID file to sign design elements of the supplied template, do not use the server ID or administrator's personal ID because this might cause problems when the application is moved to other servers or if the name of the administrator changes.

Production

Once the application is deployed to the production system, the regular production maintenance routine starts. Although the majority of work now is in control of a Domino administrator, developers are required to manage, for example, the performance impact of the growing Domino applications once they are deployed.

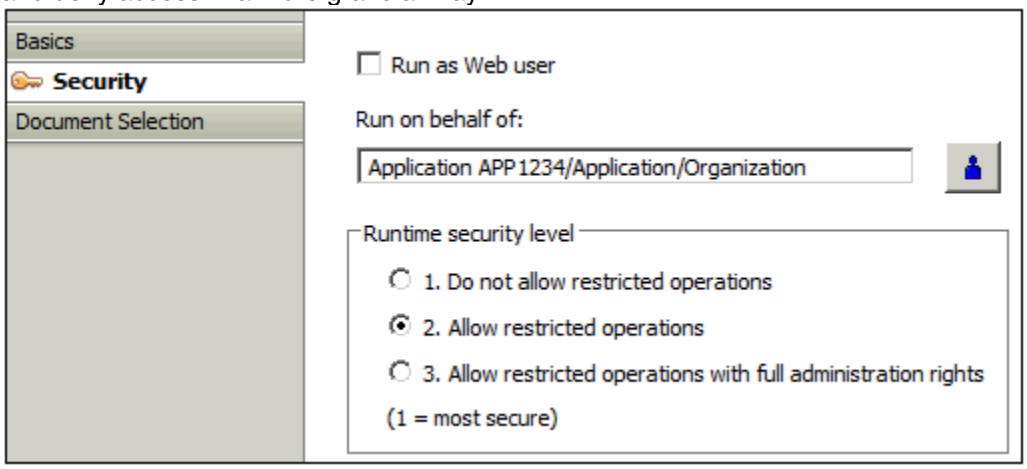
Access rights management

In an efficient Domino environment, access rights management does not require administrators to perform any changes because application owners are able to grant and deny access to users by modifying groups in the Domino Directory. You should document the process for the users to find this information easily, for example, by establishing a wiki that explains how owners can manage access rights by themselves.

Consider these good practices for access rights management:

- Manager and designer access are exclusive to administrators and servers.
  No user and no developer should have access rights higher than Editor in a production environment. Changes in existing applications should be performed by the Domino administrator by handing over design templates to the Domino administrator. This process should be agreed and documented within your organization.
- Run scheduled agents that require restricted or unrestricted functions using "On behalf of" with an application ID,
  This ID does not need to be a real Notes ID, it is just a name that you will use to grant access to resources that the agent has to access. This method allows you to grant

and deny access in a more granular way.



Availability

Availability of an application includes more than the ability to open the application as such. The application must function as defined in the requirements and must work as expected by users. The term availability is often misunderstood as end-to-end availability, anywhere, anytime. This type of availability is often beyond the management ability of the administrators and developers because they only have a subset of the environment in control. For an end-to-end availability, other components such as local and wide area network, user device, and so on are of importance. Therefore, if end-to-end availability is a requirements, the application should be designed to meet this criteria by including a broader vision on infrastructure.

In order to improve availability of a Domino application, the following items should be evaluated:

- Utilize Domino clustering to provide high availability.
  This requires an application to be built for clustering, which means developers have to avoid using, for example, references to individual server names or using unique numbering systems that are computed on one server only.
- Decentralizing Domino application servers.
  While consolidation and centralization is good for mail servers, Domino applications work very well in a decentralized environment. Local application servers can provide users who are in the locations with poor network connections the performance and application availability they require.
- Separating applications with low performance needs from those with high performance needs.
  This is where the classification of applications becomes important. With this classification, the administrators can host a large number of small applications on one server, while a different server is hosting a low number of applications with high performance needs.

Monitoring

Once an application is established in an environment, it should be monitored. You should separate the infrastructure monitoring from the application monitoring. For infrastructure monitoring, refer to the IBM Redbooks publication *Optimizing Domino Administration*, Chapter 3.1:

http://www-10.lotus.com/ldd/dominowiki.nsf/dx/Table_of_Contents_Optimizing_Lotus_Domino_Administration

While in some environments, it is acceptable to monitor the infrastructure and Domino server layer only, the complex environments with business critical applications might have certain applications that require more attention. However, due to the large variety of Domino applications and its functionality, it is not possible to predefine what should be monitored within an application. This list will give a short overview if what can be monitored:

- Usage tendency: For example, number of read and write activities, FTIndex usage,
- Growth statistics: For example, number of documents in the database, number of custom views and folders, size of documents and the attachments within
- Efficiency: For example, Agent runtime, time to render a specific page number.

You can monitor these triggers from within the application itself by custom developed monitoring agents that alert the application operators in case of any abnormal or can be part of an existing application monitoring system for example, Domino Domain Monitoring (DDM) or Tivoli ITCAM for applications.

Virus scanning

Virus scanning is important in every environment regardless of its size, however, individual settings and practices might cause drawbacks on system health or performance.

A common approach is to enable virus scanning on both the server and the workstation, while each of them focus on different items. Serious performance issues can occur if virus scanners on operating system level are configured wrong. Since Notes and Domino are accessing and modifying a large number of files on the disks at the same time, local virus scanners will interrupt read and write access at every time. Therefore, the disk I/O throughput for the server and the client will reduce dramatically resulting in a performance issue.

Here are a few examples to outline which anti virus settings should be used:

- Operating system virus scanner shall be configured to skip any Notes or Domino data.
- Application specific virus scanners shall be used to scan Domino data,
- Scanning applications on access will ensure that a document or attachment that is being opened by a user does not contain any virus.
- Virus scanners shall not modify documents stored in applications, although it might sound like reasonable for cleaning a virus. This will cause replication conflict problems when an application is distributed to multiple servers.

Backup and recovery

A Lotus Domino server backup can not be handled like a file server because Domino is keeping a large number of files opened and is performing background actions against those open files even if there is no user accessing them. Relying on normal backup software that is not aware of how to handle open files can be dangerous, as this software might claim a file as locked when Domino is trying to access. This will cause the “This database is currently being used by someone else...” error in Domino.

For more information about Domino server backup and recovery, refer to http://www-10.lotus.com/ldd/dominowiki.nsf/dx/3.9_Backup_a_Domino_Environment.

Remember the following

- Define and understand the backup policy of your corporation. Certain data might be required legally to be kept for longer than other data, so an individual backup plan for an application might be required while other applications can use the default server backup routine.
- Include the Domino Attachment and Object Services (DAOS) repository in your backup and recovery plan.
- Perform regular tests against the backup infrastructure to make sure you can restore the data that was backed up.
  For example, perform a full restore to an empty server once per year.
- Define your application restore scenario in written form, and make sure the administrators have an appropriate understanding of specific actions that must or must not be performed. for example, some applications contain scheduled agents that run by schedule to send mails. When restoring an application to a different server, you might not want those mails to be sent. Therefore, define what needs to be done to avoid this.

For additional information on how to efficiently restore an application, refer to http://www-10.lotus.com/ldd/dominowiki.nsf/dx/3.10_Restore

2.8. Upgrade

2.8.1 Upgrading applications

When it comes time to upgrade your application, there are things you, as a developer, can do to make the upgrade process easier. Here are some best practices for upgrading your Domino applications.

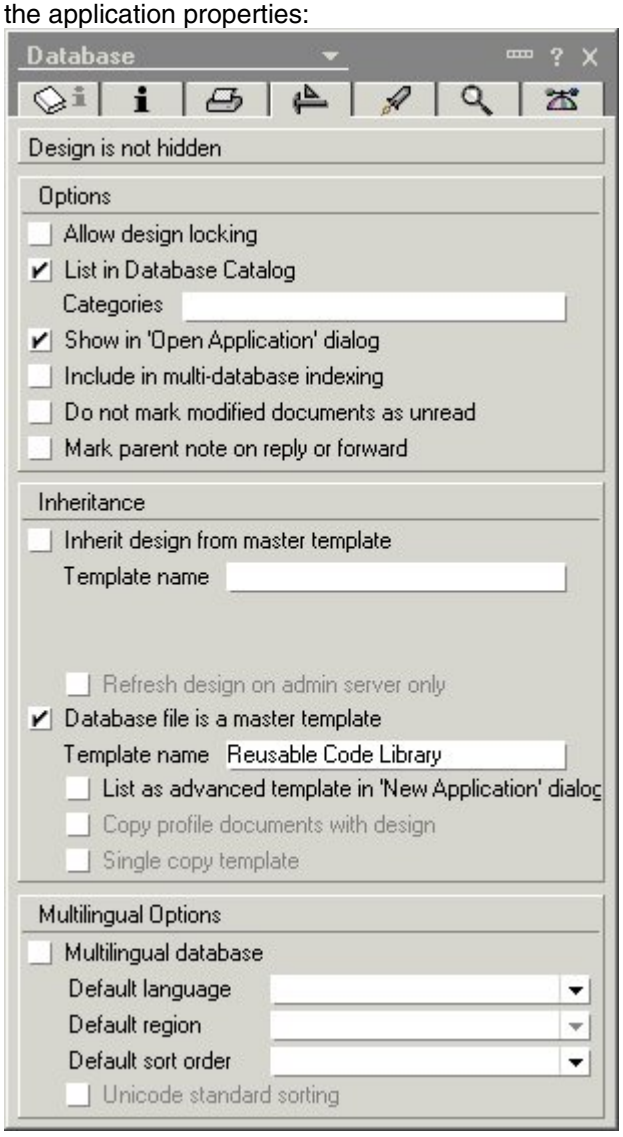
Using templates

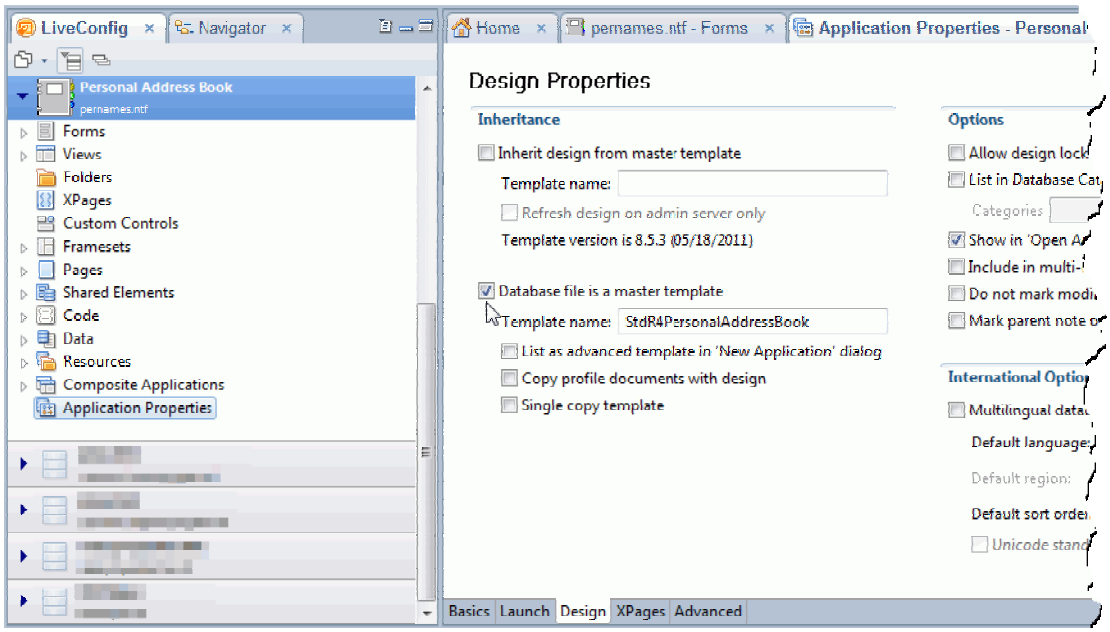
An NSF file generally contains both the data of the application and the design used to manage that data. However, to avoid problems in managing the maintenance of your application, it's better if you manage the data and the design in separate files, so that you can edit the design, and fully test the changes, without affecting the application that's in actual use. In addition, it is often the case that a design may be duplicated into multiple NSF files which contain different data. One obvious example is the user mail file, where each user has a copy of the mail file design, but it has their own data in it.

Templates provide a way to edit the design centrally and away from the application data. Once you are satisfied your changes are correct, you can apply the updated design to application databases, either directly or by placing an updated copy of the template on your servers, where the nightly DESIGN task will update applications.

Domino templates generally have a NTF file extension, but the true thing that makes a Domino application a template is the “Is A Template” setting found on the design tab of

the application properties:

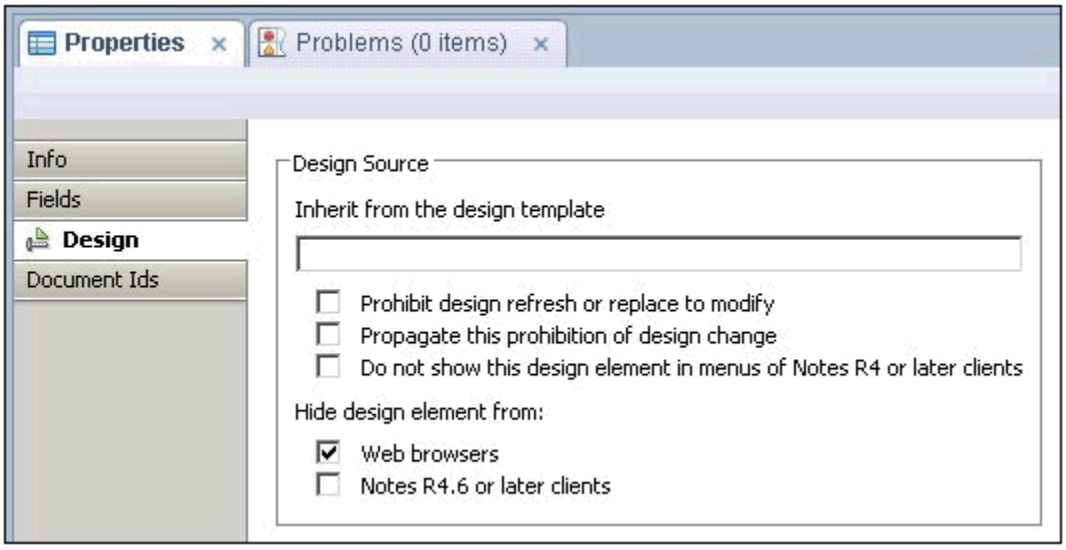




Any application that has the same value in the “Inherit from Template” setting in application properties inherits design elements from this template whenever the server's Design task runs. By default, the Design task runs at 1:00 AM daily, however, this can be changed (or even disabled) by a Domino Administrator.

Individual design element inheritance

Individual design elements can be marked to inherit their design from a specific template, or be set to not inherit from any template. These are done on the properties of the individual design element.



You can set this option on design elements an an application. As a general rule, this is only a good idea when the design contains elements that are supposed to be customized to the occurrence of the application -- they're supposed to be different from the template. With the exception of personal views and folders (such as in mail files), or customized

graphics or stylesheets that are different in each instantiation of the design, this is rare. In nearly all cases, it's best to manage the design of the application by editing the template, and making the application design identical to the template design. In cases where you have two designs that share some design elements, the template for one application can inherit its design from another template, with certain design elements exempted by checking their "Prohibit design refresh..." checkbox. Or if two templates have just a few elements in common, one can inherit only specific elements by adding the source template name in the "Inherit from..." field.

The danger of working in this way is that if part of your design changes without your awareness, because someone edited the template from which you inherit, it can break your application. Design elements are often interrelated; for instance, a form might contain a field formula that uses @DbLookup to find a key in a view, and read a value from a certain column. If someone changes the view, you may now be reading from the wrong column. You would definitely, at least, want to not accept updates automatically, but first test the changes to make sure there's no problem introduced. You may find it better to simply copy the design you want to customize to a new template with a different name, with no inheritance, and use other means to track reusable design elements.

Use special caution in allowing automatic inheritance of code-based design elements. For any code, there's a danger that someone will change the design in a way that will make it fail when called by existing code -- for instance, by inserting an argument in the middle of a function list, so that existing calls to that function are no longer correct. LotusScript code has the additional limitation that it is compiled in the context of existing code that it calls in the same application. If a script library is modified without recompiling the other LotusScript code that calls the library, the library can fail to load even if the profiles of the functions it makes calls to have not changed. There's no search that will identify all the places where a particular library is used across all applications. Therefore, it's generally better to allow the maintainers of different applications to determine when and if they will incorporate a new version of a common library into their designs. Allowing these to inherit automatically across the organization will almost certainly cause applications to fail in the field.

Generally speaking, it's a bad idea to control inheritance of individual design elements in an NSF file, since that's means you're assembling the final design for the first time in your production environment, without ever having tested it. If anything is to inherit individual design elements, it should be a template. Use the template to create test NSFs to try out upcoming changes before you release them to users.

2.8.2 Putting changes into production

When you are changing the production design of your application, there are general rules to follow to make sure the deployment works smoothly.

Back up existing design

First, you should make a back up copy of the current production template. You can revert back to that design in the case of a failure or unforeseen issue.

Document data migration procedures

If data needs to be migrated because of the new design (adding a new field that your new design requires, for example), then you should include data migration agents to update

the data, and document the steps required to run the agents. You should also work with your Domino Administrator to make sure that the data migration happens soon after the design is upgraded in order to decrease the potential for end user errors from using the new design with the old data.

If possible, design your new version to work acceptably with the old style data as well as the new, so that no migration process is needed. If you cannot find a way to do this, you definitely don't want to allow the Design task to update that design automatically.

Document data reversion procedures

For every data migration agent you have, you should also have a data reversion agent that will “undo” the migration in the case of a failure. If you have to revert back to the original design, you need a way to revert back to the original data. The agents that need to be run in the case of a reversion have to be documented.

Verification of success

After upgrading the application and migrating the data, you should have one or more users verify the new functionality of the application in production. Users should have already validated that the functionality does what they asked in a development environment, however, this is making sure the design elements were properly refreshed and the data properly migrated in production.

2.8.3 Deployment steps

You should check with your Domino Administrator for the exact steps your company uses to deploy changes to Domino applications. If the Design task has been disabled on the server, the design of the production application must be manually refreshed after deploying an updated template. If templates are not allowed to be stored on the production servers, someone with the proper authority and access have to manually refresh the design of the production application from a local template. Every company may have slightly different procedures for upgrading applications, but those procedures should be documented by the Domino Administration team.

2.9. Retirement

The Domino application portfolio of an enterprise can grow into a huge number of applications over time, deployed to leverage the existing Domino infrastructure or use of rapid application development (RAD) environment to build new capabilities to support the growing business. The document repositories and discussion databases can increase rapidly if applications are created across regions without proper planning and governance. Applications added to the Domino application portfolio without central monitoring can possibly lead to inefficiency in managing the Domino infrastructure due to redundancy, dormancy, under-utilization, and expiration. The process of application retirement or application sunset is a good practice to keep Domino infrastructure lean and efficient. The additional benefit is the recovery of disk space and extra processing power for Domino server to cater the reduced number of application databases.

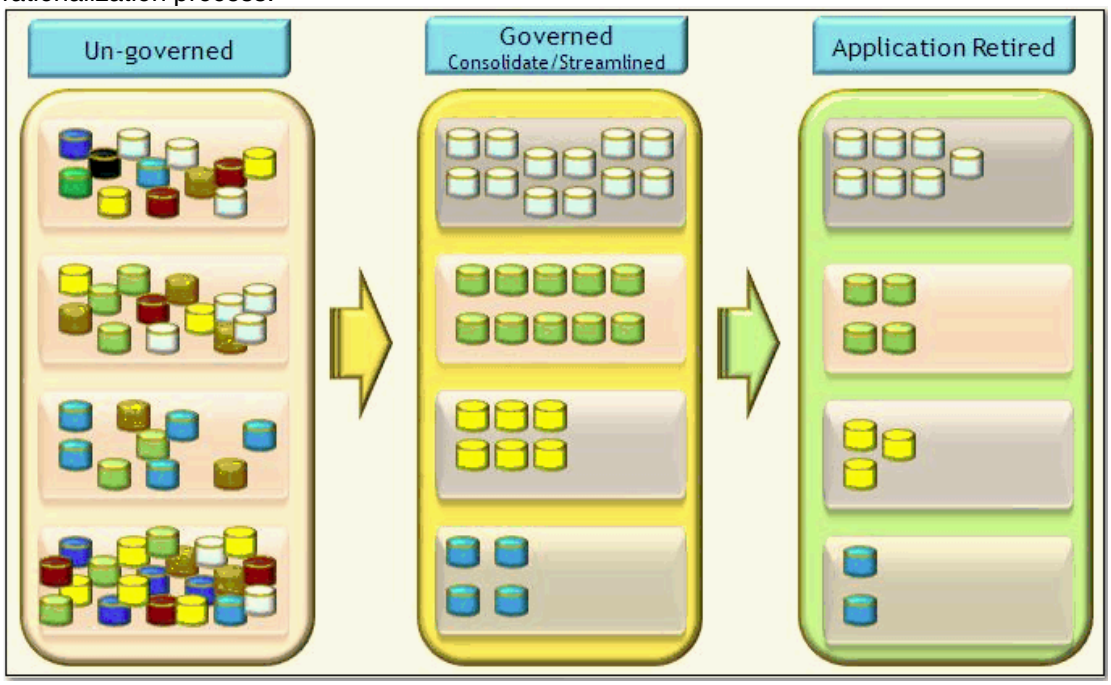
The application retirement is an essential step in the Domino Application lifecycle. The application retirement process helps identify the applications that are under-utilized or dormant. The unused applications then can be deleted or archived. The application

retirement process can also result in an adaption of new review practices as well as help achieve modernizing the existing under-utilized applications. The usage of legacy applications drops over the period of time because of their out-of-date technologies and not very intuitive features. Replacing these old-approach applications with the new XPages enabled Web2.0 capabilities and rich features that are much faster to build and use helps in reviving the application usage. The application retirement helps organization maintain a healthy, lean, and efficient application portfolio.

For the application retirement process, have a central application repository that maintains the records of existing applications, categorized under respective portfolio, for example Quality, Sales, Marketing, Innovation, Business Intelligence, Customer Relation Management, Business Support Services, Document Management, Legal, and Discussion database. Consider forming a Domino application portfolio management committee or team to ally processes and manage central application portfolio repository. Any request for new database should go through the Domino application portfolio management process. Implement an application portfolio intelligence process built with capabilities of automated report, agents, or robots. The application portfolio management process does the following:

- Prepares inventory of all applications hosted on all servers across various locations.
- Track user activities or traffic for each application in the inventory. you could re-use the information in database catalog on each server and database usage statistics.
- Generate reports about dormant, under-utilized, expired, ready for renewal, retired application and decommissioned databases.

This process helps your team to rationalize applications, their retirement, decommission, and archive to avoid redundancy and increase common share (that is to reuse existing database for similar purpose, generally the document repositories and team discussion databases). This process also helps to bring better governance, visibility, and control on the databases across Domino application portfolios at different locations. The following figure shows the state of ungoverned, governed, and retired applications, resulting from a rationalization process.



Chapter 3. Infrastructure impact for application development

This chapter discuss how the infrastructure such as severs, operating systems, Domino versions can impact the Domino application development.

3.1. Using development, QA, and production servers

3.1.1 Domino environment

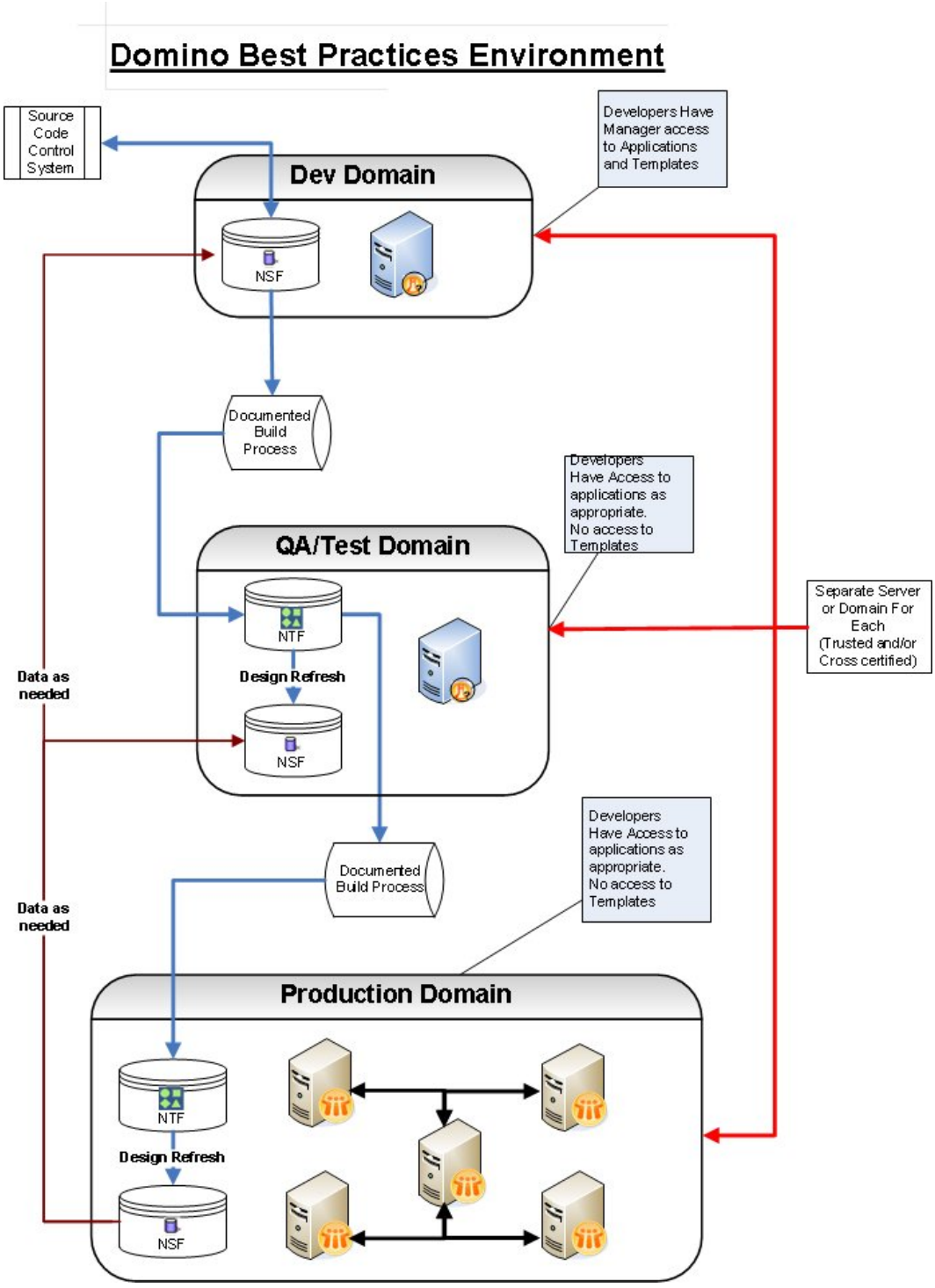
Domino provides a rapid application development environment, this does not mean that one should ignore standard development practices. There should be at least a separate production environment and development server, even if there is only one developer.

The normal Domino environment consists of a development domain, user testing or quality assurance (QA) domain, and the production domain . The access rights a developer has changes with each domain.

The ultimate environment has the development domain and the QA domain be an identical configuration as the production domain. This in reality is cost prohibitive, so the next best setup is to have the QA domain match the configurations in production that would likely cause issues. If in production you have internal application server and servers with web access that are available to third party over the internet hosted on a secured access zone or a demilitarized zone (DMZ), then the QA domain should model that structure. The idea is to be able to test the application fully so it will not have issues when moved to production. At this point you are trying to match the overall configuration of production, not that actual setup. If you access a relational database management system (RDBMS) in production, then you must have the same access in QA. If you have five application servers in production, it does not mean that you need five for QA.

Because you do not hard code server names into the application, as that would be a really bad practice, developing your application on a different server is not an issue. If you have code that accesses another database on a different server, then the target server's name should be on a profile or configuration document that can be easily changed. There is an application on OpenNTF.org that can help you maintain these configuration documents if you have a large number of applications that interact. The application even supports failover when a server is down.

This figure shows what the Domino environment looks like from the server prospective.



The management of the flow of the code between the areas can range from a simple Notes database to utilizing a third party tool to control the templates. What is the most important rule is that you have a procedure and follow it.

3.1.2 Development environment

The development server can be anything from a single virtual machine (VM) running on the developers workstation up to multiple servers. This is the area for developers to build and test the applications. If you are doing team development then the Domino development server has to be on hardware that can be accessed by all developers. We discuss virtual machines and how they can be used to help with development and testing in 3.2 Multiple version support with virtual machine.

For the development environment, it should be set up to allow the developers to easily change the configuration to assist with developing the applications:

- One or more developers should have the basic Domino Administration skills.
- Developers have manager access to server and applications.
- Developers use the Lotus Notes Administration Client to:
    - See view information.
    - Use server console to check for errors.
- Developer must be able to create databases and replicas on the server.
- Developers can change Domino Directory documents such as the Internet Site documents.

The server configuration should have:

- No connection docs unless there are multiple development servers.
- Mail routing turned off, you can access the mail.box to see what messages look like. User can get confused by notifications from the development server.
- Developers have full access to the server's mail.box.
- If the production servers use SSL, then development should have at the minimum a Domino self-certificate. With the cost of SSL certificate running under $100, it is cheaper to purchase one then have to deal with the administration of the self-certificate.

The server can be in the same Domino domain as production or in its own domain:

- If in the same domain:
    - The Domino Directory is a copy of the main production domain so no chance of replication or It has its own copy and the production Domino Directory is set as a secondary address book using Directory Assistance.
    - Give the development server a different Notes Named Network, so users do not see it.
- If in a different domain:
    - The main production Domino Directory is set as a secondary address book using Directory Assistance.
    - Cross certify the development domain to the production domain to make it easier for everyone.

3.1.3 QA and user testing environment

Developers should have, at the most, the designer access if they are allowed to make minor code changes to QA, otherwise, they have the same access as they do to production.

The Domino QA server configuration should match as closely to production as possible. If you have an internal Domino application server and a separate Domino server for web access in production, then you should have the same in QA.

The server can be in the same Domino domain as production or in its own domain:

- If in the same domain:
    - The Domino Directory is a copy of main production domain so there is no chance of replication; or It has its own copy and the production Domino Directory is set as a secondary address book using Directory Assistance.
    - Give the development server a different Notes Named Network, so users do not see it.
- If in a different domain:
    - The main production Domino Directory is set as a secondary address book using Directory Assistance.
    - Cross certify the development domain to the production domain to make it easier for everyone.

3.1.4 The build process

Use templates to move the design between environments.

Development to QA

The steps a developer takes to promote an database from development to QA are as follows:

1. Create a template from the development database.
2. On the template database properties, Select **Database file is a master template** and enter the template name.

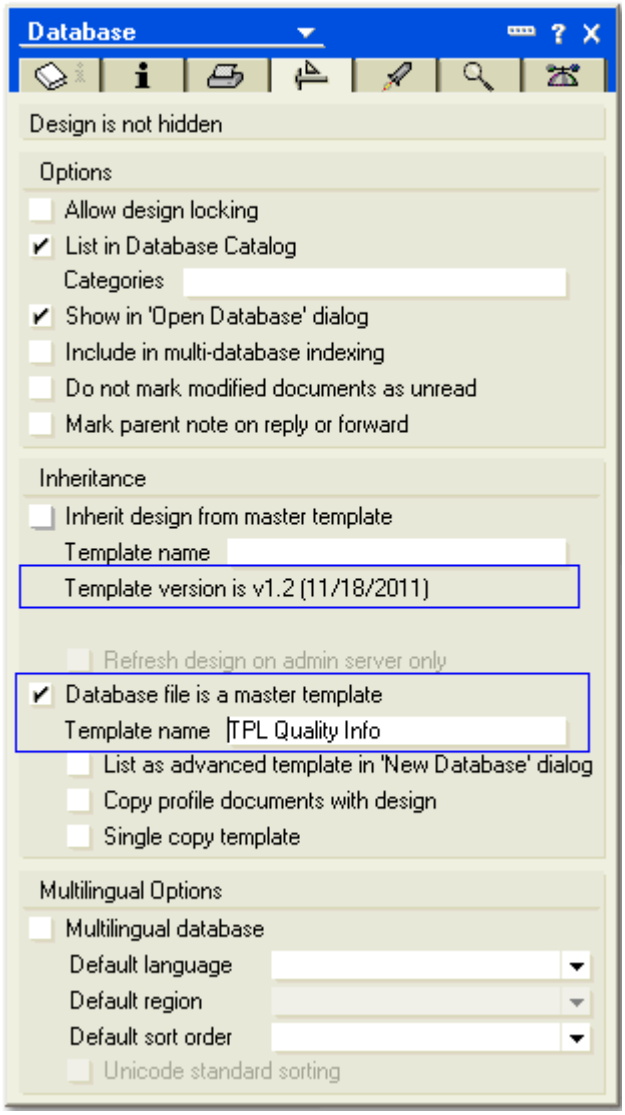
There's no fixed rule as to whether the template name should include a version number (e.g. *TPL Quality Info v1.2*). Usually, applications undergo continuing maintenance. If your intention for the application is that every instance of the application will have the same design, receiving updates from a single common template, it's better to not put a version number in the template name. That would just make it harder to maintain the design of the application across all instances.

However, if you intend that there might be different versions of the same application on the same server, it may make sense to include the version number in the template name. This might be the case if you're developing your template as a product to sell, for instance. Different customers might deploy different versions within the same company, and you wouldn't want to have older versions automatically upgraded by the DESIGN task because someone installed a newer version of the template on the server -- especially if there are data updates that should happen at the same time as the upgrade, or if the users of the application might have older clients that don't support some features used by your new version.

If you do put version numbers in your template names, make sure the different versions of your template are not replicas of each other. This can lead to older version templates being overwritten by the newer version through replication, and occasionally can result in duplicated design elements in both replicas (if the design elements with the same name

didn't have the same universal ID). This causes a huge mess.

If you don't want to put the version number in the template name, but you still want to have the information stored somewhere so that you can easily tell whether a particular NSF has an up to date design, you can add the template version as non-editable information in the database property dialog, as shown below.



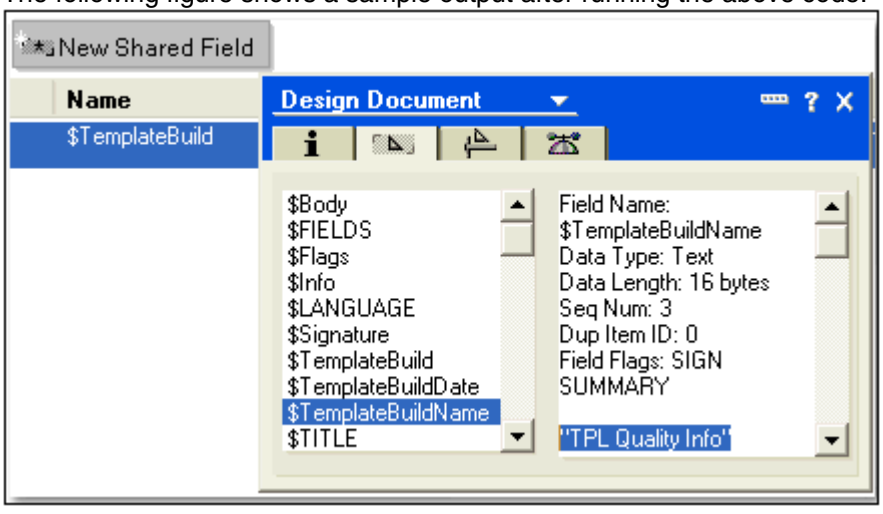
This works in Notes/Domino 6.0.1 and later, and is done by creating an editable shared field named “$TemplateBuild”. This is a special field name recognized by the database properties code; the shared field design element contains three extra items not in a normal shared field; the items are named $TemplateBuild, $TemplateBuildName and $TemplateBuildDate. You can assign values to these extra items by a LotusScript or Java agent.

The Notes/Domino 6 discussion forum contains sample LotusScript code, written by Thomas Gumz, for retrieving the version information in the $TemplateBuild element. The following code snippet is a modification to assign the values of these parameters in the template. It assumes that it is running in the same template that is to be modified; of

course, you can also modify this code to operate on another database whose path is supplied as an argument the the constructor of NotesDatabase.

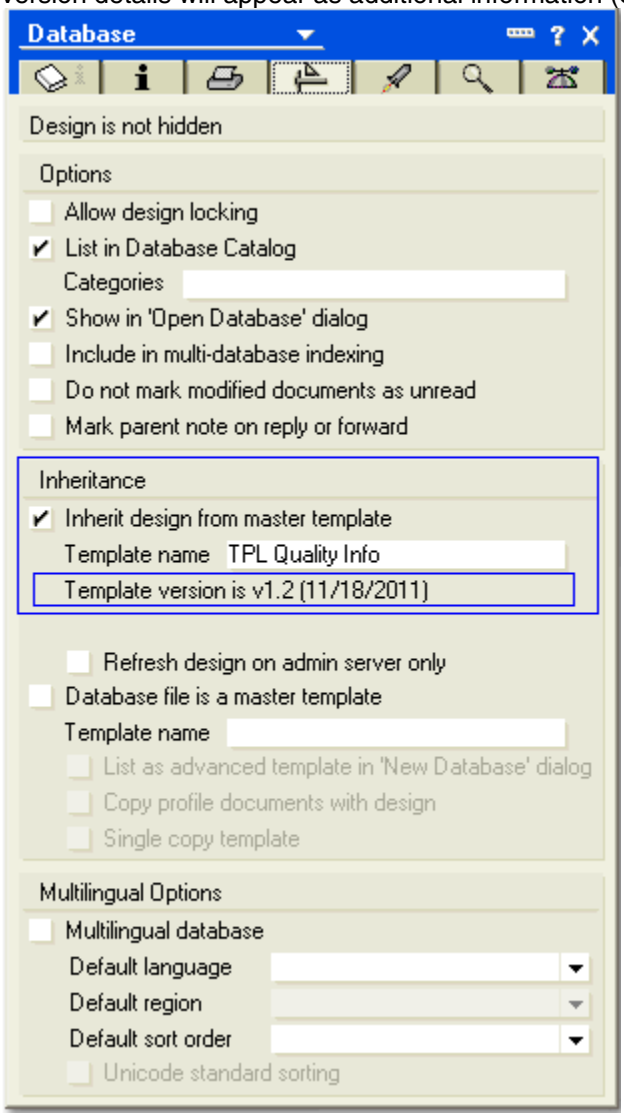
```
'....Author: Thomas Gumz
Dim session As NotesSession
Dim db As NotesDatabase
Dim nc As NotesNoteCollection
Dim doc As NotesDocument
Dim iCount As Integer
Dim sNoteID As String
Const ITEM_BUILD = "$TemplateBuild" '.....Modified
Const ITEM_BUILD_NAME = "$TemplateBuildName" '.....Modified
Const ITEM_BUILD_DATE = "$TemplateBuildDate"
Set session = New NotesSession
Set db = session.CurrentDatabase
Set nc = db.CreateNoteCollection(False)
If Not nc Is Nothing Then
nc.SelectSharedFields = True
Call nc.BuildCollection
sNoteID = nc.GetFirstNoteId
For iCount = 1 To nc.Count
Set doc = db.GetDocumentByID(sNoteID)
If Not doc Is Nothing Then
If doc.HasItem(ITEM_BUILD) Then
'.... Msgbox "Version: " & doc.GetItemValue(ITEM_BUILD_NAME)(0)
Call doc.ReplaceItemValue( ITEM_BUILD, "v1.2" ) '.....Modified
Call doc.ReplaceItemValue( ITEM_BUILD_NAME, "TPL Quality Info" )
'.....Modified
Call doc.ReplaceItemValue( ITEM_BUILD_DATE, Now() ) '.....Modified
'.... Msgbox "Date: " & doc.GetItemValue(ITEM_BUILD_DATE)(0)
Call doc.Save(True, False)
Exit For
End If
End If
sNoteID = nc.GetNextNoteID(sNoteID)
Next iCount
```

The following figure shows a sample output after running the above code:



1. Provide the template and the related information such as the agents that have to clean up the data, fixes, and so on to the QA Administrator.

2. The QA administrator then perform the following:
    a. Copy the template on the QA server.
    b. Sign the template using the server ID, their ID, or a test ID.
    c. In the QA environment, enable the target database property of inheritance to inherit from the master template. Then refresh the design of this target database in the QA environment. On successful refreshing the template, the version details will appear as additional information (See the figure below).



    d. Notify the QA team that the application is ready for testing.

Let us review how Domino design refresh works: When you select a Domino application and select **File -> Database -> Refresh Design**, Domino retrieves the database's template name found on the design tab of the database properties box and then locates the template with that master template name. Domino matches up design elements between the application and the template, deciding which are to be updated, added or deleted. This matching algorithm is fairly complicated, using the universal ID of the design notes, the value in the $TITLE item (particularly the last value, the alias), the language setting if this is a multilingual application, and the Notes/web/modile setting. These last two are used in case there are design elements with the same name but different languages, or designed for different clients. Generally speaking, the details of

how the match is done aren't important, but if it seems like design replace/refresh isn't working as you expected, consider the possibility that by renaming a design element, you have made the design replace process not realize that it's "the same" design element as before.

Testing the application

For testing, you must have test IDs that represent the different levels of access that user can have with the application. If you have users the are reader and others can edit, then you need test IDs with the same access level. Domino has the ability to hide and control access to everything from the application down to the individual design elements.

The application has to be tested not only for correct functionality but also using the client application that the end user will be utilizing. Different version of the Notes clients do not have as much issues as different version of browsers have. Unless this is for internal use and everyone has the same browser, you have to be able to test the application with a variety of browsers and versions. There is a few open source tools that can be used for testing web based applications. The benefit of using these are that you can test overall functionality very easily and be sure there are no regression issues. Your changes to one form may cause another form or code to now malfunction.

Testing the entire application for each release can be very time consuming. That is why it is important to document the changes made in development and build a test case for those changes. The test case will indicate what functionality the users should see. It should correspond to the original change or design request. An example is adding a new form for the user, the design requirements will list the fields and the expected workflow. The test case would be the steps the user will take to utilize the form and the expected out come.

Especially for new applications, performance testing under realistic conditions of server load and number of documents is very important. We very commonly see applications that worked fine with a few dozen test documents and one user, but once they get tens of thousands of documents, their performance suffers badly. These problems can be addressed by following the design principles in the whitepaper "Performance basics for IBM Lotus Notes developers."

QA to production

When the application has been *fully* tested, there should have an approval process to indicate it is ready for production deployment. Provide the template and related information such as the agents that have to clean up the data, fixes, and so on, to the production administrator. The Production administrator then do the following:
1. Sign the template using the server's ID or the administrator ID used for signing.
2. Apply the QA template to the appropriate Domino database.
3. Run any update agents that are needed.
4. Enable all scheduled agents.
5. Notify users of updates

3.2. Multiple version support with virtual machine

3.2.1 What is a virtual machine

A virtual machine (VM) is a software application that allows you to have a simulated computer that executes programs like a physical machine. This lets you test applications in an controlled environment. You can configure the hardware that the virtual image thinks it has access to. The benefit is that you have a large file instead of having an actual computer. This allows you to easily back it up or to copy it to different physical computers. Some of the versions of virtual machine provide for versioning, cloning, and linking of virtual machine images, to make it easier to create and use. With a virtual machine, the computer the virtual machine application is running on is considered the host and the virtual machines are the guests. One of the criteria in selecting a virtual machine application for development is which Operating Systems are needed for testing and which Operating Systems are you running on the host system. The selection of the guest's Operating System is sometimes restricted, also some are restricted to run on certain host Operating System.

Some of the virtual machine applications available allow the user of multiple virtual machines running on the same computer. This allows you to have multiple virtual machines running at the same time on your computer.

There are versions of virtual machine application that include their own Operating System, so they do not run on a host Operating System. This provides for less overhead which results in better performance. These are used in server consolidation since they are best for performance.

The main advantages of virtual machines are:

- Multiple Operating System environments can coexist on the same computer, in complete isolation from each other.
- You can test applications, maintenance, high availability, and disaster recovery.

The main disadvantages of virtual machines are:

- The virtual machine is less efficient than a real computer as it accesses the hardware indirectly.
- When multiple virtual machines are concurrently running on the same physical host, each virtual machine may exhibit varying performance, which highly depends on the workload imposed on the system by other virtual machines.
- They share the memory and storage of the host and are therefore limited to how much they can utilize.
- The reasons for using a virtual machine include:
- Develop and test multiple Domino and Notes versions on a single personal computer.
- Connect virtual machines to simulate and test multi-server configurations.
- Use multiple snapshots to facilitate testing.
- Archive test environments on file servers where they can be easily restored and shared.
- Less need to keep buying hardware or to clean machines.
- The virtual machine can be configure to "roll-back" all changes when shutdown.
- Test older Domino Server versions within virtual machines.
- Crash a server and not impact other developers or your personal computer.
- Test how applications perform with poor network connectivity.
- Testing Domino applications running on different Operating Systems.

- Testing a cluster's failover impact to your application.
- Test applications using different browser versions.

The following are some of the currently available virtual machine applications:

- VirtualBox is an open source virtual machine developed by Sun Microsystems (now owned by Oracle Corporation). Works extremely nice on Ubuntu workstations.
- VMware provides virtual machines for x86.
- Xen virtualization system is intended to run a moderate number of full-featured operating systems, Linux, Solaris, as well as Windows XP and 2003 Server (requires Version. 3.0 and an Intel VT-x(Vanderpool) or AMD-V (Pacifica)-capable CPU).
- KVM is a Linux kernel module maintained by Red Hat, that enables a modified QEMU program to use hardware virtualization.
- Virtual PC from Microsoft allows Windows 7 to run virtual Windows XP, Windows Vista, and Windows 7 as guests.
- Hyper-V from Microsoft runs on Server 2008 R2.
- Parallels Workstation supported host Operating System are Mac, Windows, and Linux.

3.2.2 Virtualization best practices

When use virtual machine for Domino application, consider the following. Note that some of the recommendations might not be valid for every virtual machine application.

- Allocate fixed memory size and fixed disk size.
- If possible, run all the virtual machines RAM in real RAM, avoid swapping.
- Do not limit the RAM the virtual machine can effectively use .
- Lotus Domino calculates its cache size based on Operating System available memory. It is not aware of artificial limitations imposed by virtual machines.
  In Memory -> Resources, the “Unlimited” really means “up to the amount allocated to the virtual machine”, therefore, is the correct setting.
- Do not share volumes.
- Pre-allocate disk space and install on a recently defragmented drive.
- Use virtual machine file store volumes instead of directly allocated physical volumes.
- Defragment drives - Defragment drives on both virtual machine and the host system of the virtual machine.
- Create virtual machines on a different partition to the host operating system or swap file. It makes sense to store virtual machine images on fast external USB, Firewire, or eSata drives.
- Turn off hardware devices that are not required in the virtual machine such as USB ports, Sound Devices, Floppy Disks, and CDs.
- Configure the virtual machine to start with the CD and DVD optical drive disconnected.
- Either set fixed one CPU or two to *n* CPUs.
- With low CPU demand configuring one fixed CPU should give best performance.
- Never allocate more CPUs then the host system has.
- The guest operating systems have different kernels for single or multiple and 32-bit or 64-bit CPUs.
- Use a host with a 64-bit CPU to test guest that will run on 64-bit operating systems.
- Enable hyper-threading on the host machine.

If using VMware then install virtual machine Tools.

Small machine example for Domino application server might have the following configuration:

- 32 bit
- 1 CPU
- Max 3 GB RAM (more is not effective on a 32 bit machine)

3.2.3 Snapshots

A disk snapshot is a copy of the virtual machine disk file recorded at a specific time. The snapshot preserves the original VM disk file by disabling writes to the original disk file. All the new writes are made to the snapshot version of the VM.Snapshots. Because the snapshot file is only a change log of the original virtual disk, it is not a backup and you should not rely upon it as a direct backup process. The virtual machine is running on the most current snapshot, not the original vmdk disk files.

- Every VMware snapshot creates a new delta file, while the older one becomes read-only file. Since changes to the VM are made directly to the delta file, the file size keeps increasing. Multiple snapshots of a VM should be taken only if the VM has frequent changes or in case of system level changes such as patches or driver updates.
- Allocate 30% more storage to each VMware snapshot.
- Use snapshots as a versioning tool rather than as the data backup.
- Set the snapshots retention period according to your organization's requirements and delete the out-of- date copies. A delete operation with the VMware snapshot tool commits all the changes stored in the delta file to the original VM. Therefore, plan the snapshot deletion time. Avoid bulk deletion unless the snapshots are creating storage overhead. Delete the delta files one at a time, with the oldest copy being deleted first. Bulk delete operations should be performed when I/O activity on the original VMs is at its lowest.

3.2.4 Clones

Clone a virtual machine to easily deploy identical copies of a fully configured virtual machine. You can do anything with a clone that you could with the original virtual machine. The clone may be linked or full:

- Full clone: This is an independent copy. A full clone shares nothing with the parent after the cloning operation.
- Linked clone: This is a copy of a virtual machine that shares virtual disks with the parent virtual machine in an ongoing manner. Conserves disk space.

3.2.5 Templates

The first and simplest reason for using virtual machine templates is efficiency. With templates, many repetitive installation and configuration tasks can be avoided. The result is a fully installed, ready to operate (virtual) server in less time than manual installation. Templates are also used to help enforcing consistency and standards. Deploying from templates helps to enforce corporate standards such as including anti-virus and management software in any machine connected to the network.

The use of templates is a best approach to set up a virtual machines because templates can help protecting a virtual machine image from any changes by the users. When the

virtual machine is in template mode, it cannot be deleted or added to a team, and the virtual machine's snapshots cannot be deleted.

Note: To protect the template, set the file to read only or remove write permission for the virtual machine’s files. This precaution prevents inadvertently deleting a virtual machine through the file system, or from disabling template mode for a virtual machine.

3.2.6Windows guest performance tuning

Microsoft Windows offers certain simple yet highly productive tweaks that can be universally applied to any Windows virtual machine guest operating system installation. This section describes modification of certain registry entries. The Microsoft Windows registry editor is an advanced tool and should be used by advance user. Take appropriate backup of registry before you start making changes.

Kernel paging
Disable the kernel paging of code to give a performance boost. It improves performance on machines with a lot of memory, because it prevents disk reads to get code or data, below are the registry entries to be edited:

- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management
- Value name: DisablePagingExecutive
- Data Type : REG_DWORD
- Data : 0 or 1 (Default is 0)
  Set to 1 to stop paging for “pagable” parts of kernel drivers. This can use more system memory but run faster.

I/O Tuning
This entry controls the maximum amount of RAM that can be locked for I/O operations. The default minimizes RAM usage. An I/O intensive system could benefit from larger buffer sizes.

Caution: Setting this parameter too high can result in slower performance. Set it in increments and see how it affects your system. A rule of thumb is to set to a value calculated by using the formula "Total RAM size in KBs divided by 8". Edit these registry entries to improve I/O performance:

- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management
- Value name: IOPageLockLimit,
- Data: Set to a calculated decimal value of the total RAM size in kBytes divided by 8
- Increase performance by disabling the last access update file's date.

The following changes are to set the operating system to disable last access update for directories and files on the Windows server, which translates into a radical reduction in the total I/O. Because the Domino processes access the data files frequently and have their own headers with time stamps within them, this saves I/O resources in updating time attributes for files or directories.

- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\

- Value name: NtfsDisableLastAccessUpdate.
- Data: Set it to 1 to stop updating the last access value.

3.3. Browser considerations

Lotus Domino is supported on multiple browsers. It is important that you are aware of the environment your application is supposed to work in.

Modern browsers support most of the features that you can use in an XPage application , however, you have to make sure that you cover all the pitfalls by properly testing your application in all the concerned browsers. You can also take advantage of the multiple browser support feature by asking your application to change behavior on the fly. For example, you can switch themes based on the browser user is using.

List of browsers supported by Lotus Domino 8.5.3 can be found here: https://www-304.ibm.com/support/docview.wss?uid=swg27022468

3.4. Operating system considerations

Lotus Notes and Domino are supported on multiple operating systems. It is very important that you are aware of the environment that your application is supposed to work in.
Platforms supported by Lotus Domino server are: AIX, iSeries, Linux, Solaris, and Windows.
Platforms supported by Lotus Notes are: Linux, Mac OS, and Windows.
Platform supported by Lotus Domino Designer is: Windows.

Every operating system has its own boundaries and rules which you should consider while developing your application. Some of the common factors that vary on different operating systems and can effect your application are:

- Allowed file size
- Allowed characters in names
- Memory consumption limit
- File and folder access control
- Possible scalability
- Number of connections allowed

Here are some links that can help you determine the differences between various operating systems:

- Comparison of operating systems: http://en.wikipedia.org/wiki/Comparison_of_operating_systems
- Comparison of file systems: http://en.wikipedia.org/wiki/Comparison_of_file_systems
- Known limitations of Notes and Domino 8 releases: https://www-304.ibm.com/support/docview.wss?uid=swg21308379

For multi platform applications, you should be very careful on how you write your code because you are bound to run into trouble if you even ignore minor things like how to access file system on various operating systems.

If you are using external API's then you should check if there is any dependency on the operating system in the code.

You should also make sure that you test your application properly on all the supported platforms and take into consideration both the server and client side of your applications because you will see some differences in the behavior at both ends.

Your code should be optimized in such a way that it will act keeping in mind an effortless performance on any platform your application will be running on.

3.5. Application localization

The explosion of the internet has ushered in a new era of business – one without borders on a multinational global scale. To accommodate the need to communicate effectively across language barriers, Lotus Domino gives you many options to take a native-language application (such as English) and translate it into multiple languages. These tools include Domino Global Workbench, creating individual multilingual design elements, and XPages localization, all of which are discussed below.

3.5.1 Design considerations

If you wish to deploy a Domino application in multiple languages, there are many design considerations that you must undertake. These include:

- The language base that your application will support (such as English, Spanish, and French).
- The level of expertise available to perform the translation of text between languages, either in-house or through an external translation service.
- The complexity of the application.
- The development of the application using traditional or XPages-based development environments.
- The scope of the application translation – will it be the entire application that gets translated, or just certain parts?
- The frequency of updates to the translated text within your Domino application.

Important note: Translation impacts text-based data presented to the user through the application, and does not include the translation of programming code, menus, messages, and other items therein.

Once you have a clear picture about the application itself, you have to deploy the method that is best suit your needs in maintaining a consistent application across multiple languages, using the tools available for Lotus Domino.

3.5.2 Domino Global Workbench

Domino Designer extends itself naturally through an integrated translation tool called Domino Global Workbench. This tool, when installed along side of Domino Designer, gives you the facilities to translate the native application into multiple languages.

Due to the complexity of implementing a Domino Global Workbench solution, we outline the steps here and reference you to the list of items at the end of this subsection for further information.

Before you begin, it is important to understand the steps involved in managing a multilingual Domino application using Domino Global Workbench:

1. You must first setup the application (either in .NTF or .NSF format) with the default language that you will use to construct the initial application.
2. After the application is designed, you set up a Domino Workbench glossary based on the template DWGGLOSS.NTF that is installed with Domino Global Workbench.
3. You then use the WorkBench to construct a database from the original application. This database, called a tagged database, is identical to the database of the original application.
4. Within the tagged database, each piece of text that you want to translate is given an unique identifier (ID), which is linked to an entry in the original application. Domino Global Workbench will attempt to create tags for all translatable parts of the source application that you define. If there is something that it cannot create a tag for, you can manually create the tag using the workbench.
5. You must now manually translate the tagged text into the target language using the glossary created earlier and the Notes Client.
6. Now you have the source application, the tagged database, and the glossary, you build language databases using these components. The language database is the physical application that the users will access.
7. Once the language databases are created, you can apply the updates by modifying the original application and the glossary, using Domino Global Workbench to push the changes out to the language applications.
8. In addition, you can configure the Synchronizer, a Domino server task that can apply changes to the language applications automatically.

Important note: While Domino does allow translated items to be synchronized across multiple language applications, you are still required to have the tagged text translated from the native language into the target languages.

For more information about Domino Global Workbench, please visit:

- **Domino Designer Help** (search for “Global Workbench”).
- **An Introduction to Domino Global Workbench 8.5**, located at: http://www.lotus.com/ldd/ddwiki.nsf/dx/Introducing_of_Domino_Global_WorkBench_8.5
- **Translating a Domino Application using Domino Global Workbench**, located at: http://publib.boulder.ibm.com/infocenter/domhelp/v8r0/index.jsp?topic=/com.ibm.designer.domino.main.doc/H_TRANSLATING_AN_APPLICATION_USING_DOMINO_GLOBAL_WORKBENCH_STEPS.html
- **Product Documentation** that comes with Domino Global Workbench.

Best practices

To get the most out of Domino Global Workbench, consider the following:

- Use Domino Global Workbench to translate native Domino applications into multilingual format.
- Domino Global Workbench can translate tagged text at the document level into the target language.
- Have a technical person perform the initial analysis of the native application, gathering information on what components are to be translated.
- Have a translator performs the actual language translation of the items, based on the information provided to them.
- Store the translated information in the glossary database. This allows you to apply existing translated information to other (new) applications without having to manually translate the information again.

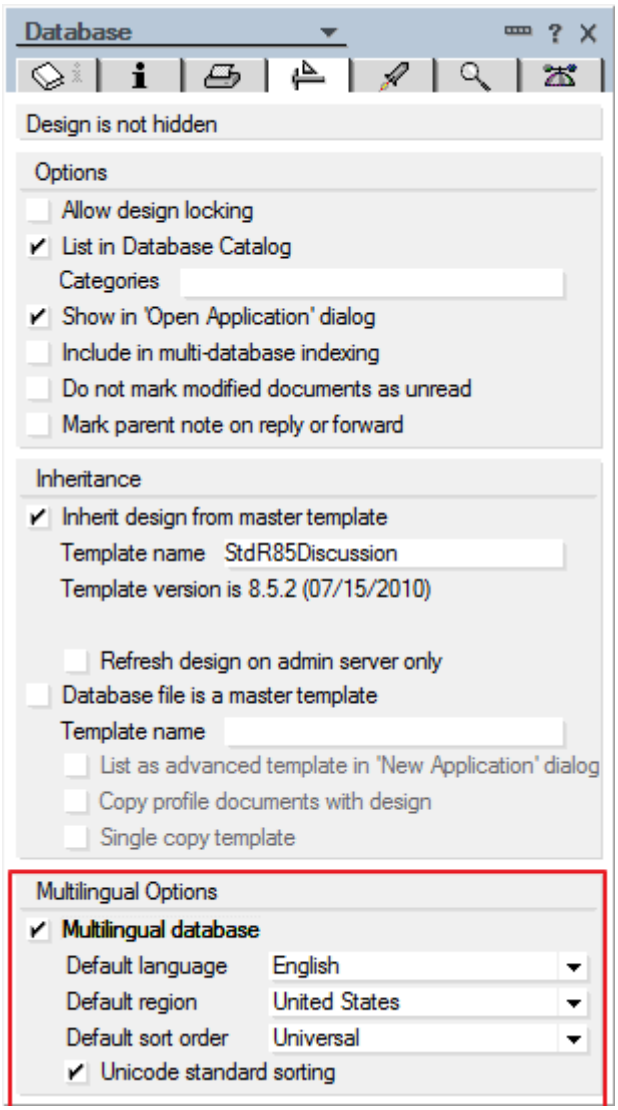
3.5.3 Design element localization

For many Domino web development efforts, it is common to use native design elements (such as forms and pages) to build web applications, opposed to document-level application development. These web applications tend to be more static in nature, with the frequency of changes being minimized.

Localizing design elements permits Domino to work in conjunction with the language settings in your web browser to determine which design element (in what language) that Domino will deliver to you.

If you plan to deploy a Domino web application that relies more on design elements than documents, you can deploy those design elements to support multiple languages in a straightforward manner, as outlined below by example.

Step 1: Configure the application as multilingual.
This is accomplished through the database properties dialog box, as shown:



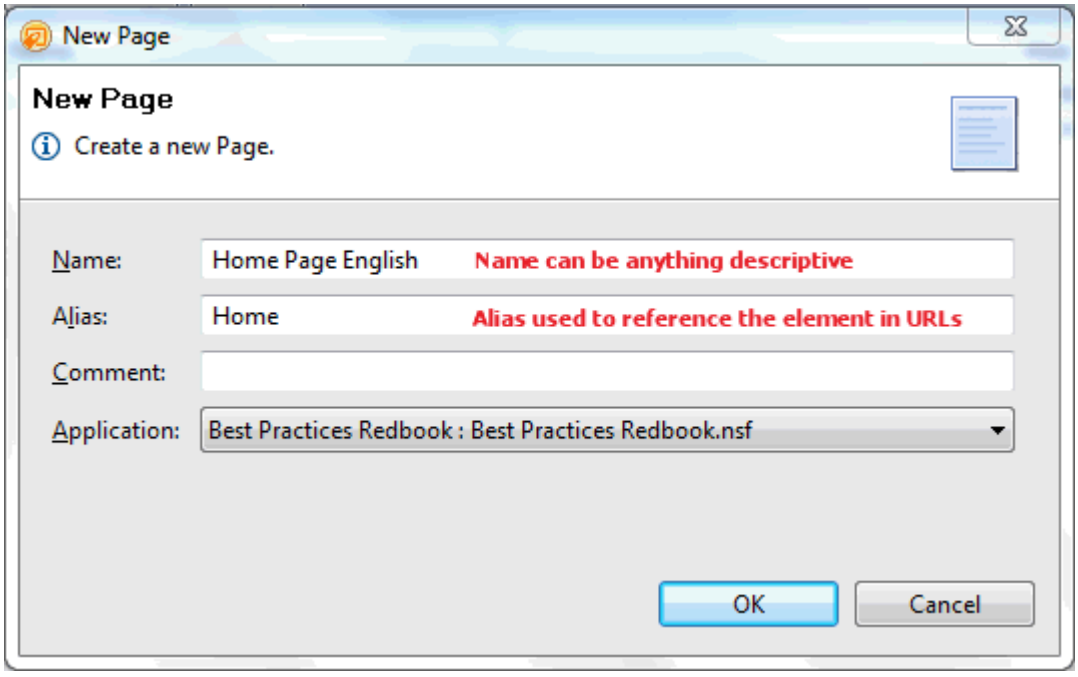
The above is establishing a baseline for your application, informing Domino that the application:

- Is multilingual.
- The default language is English.
- The default region is the United States.
- Use Unicode (Universal) sorting by default.

Step 2: Create the default language design element.
For the purposes of discussion, the following assumptions are made:

- You want to create a Domino web application.
- You will use Pages as the key design element in your development.
- You want to support the English and German languages.

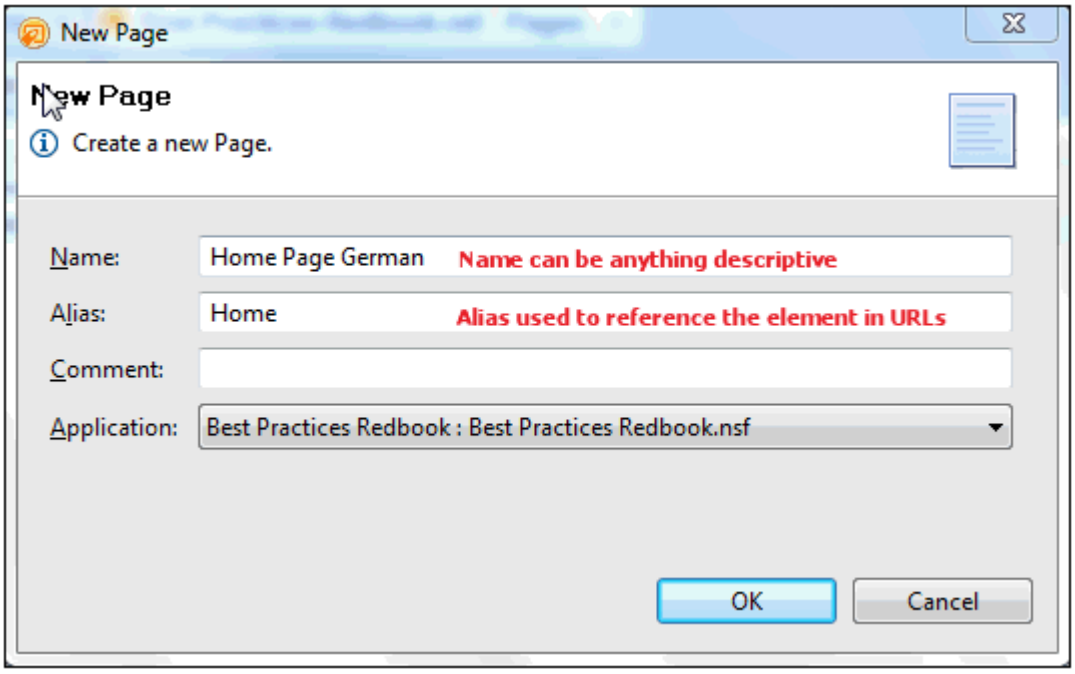
You can start by creating a new web application with a Home page utilizing the Domino Designer page design element, for English language users, as shown:



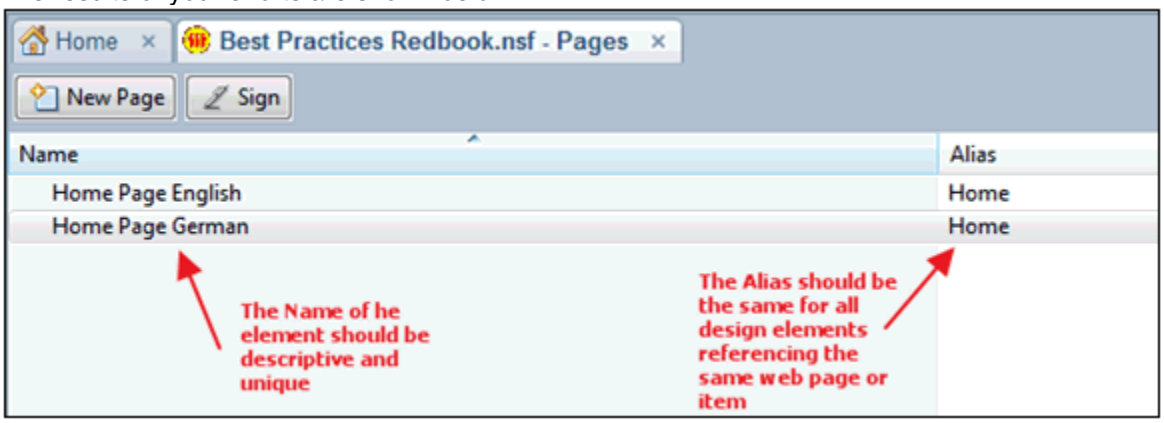
The important thing to realize is the naming convention used above:

- The design element name is always different for each design element representing the same web page in the application. As shown above, the name describes the page as being the Home Page in the English Language.
- The design element alias is always be the *same* for each design element representing the same web page in the application. As shown above, Home represents the home page, and, will be the same across all variations of the home page in different languages.

Based on our earlier assumptions, you need two home pages – one in English, and another in German. Therefore, you can create the German language web page as shown below:

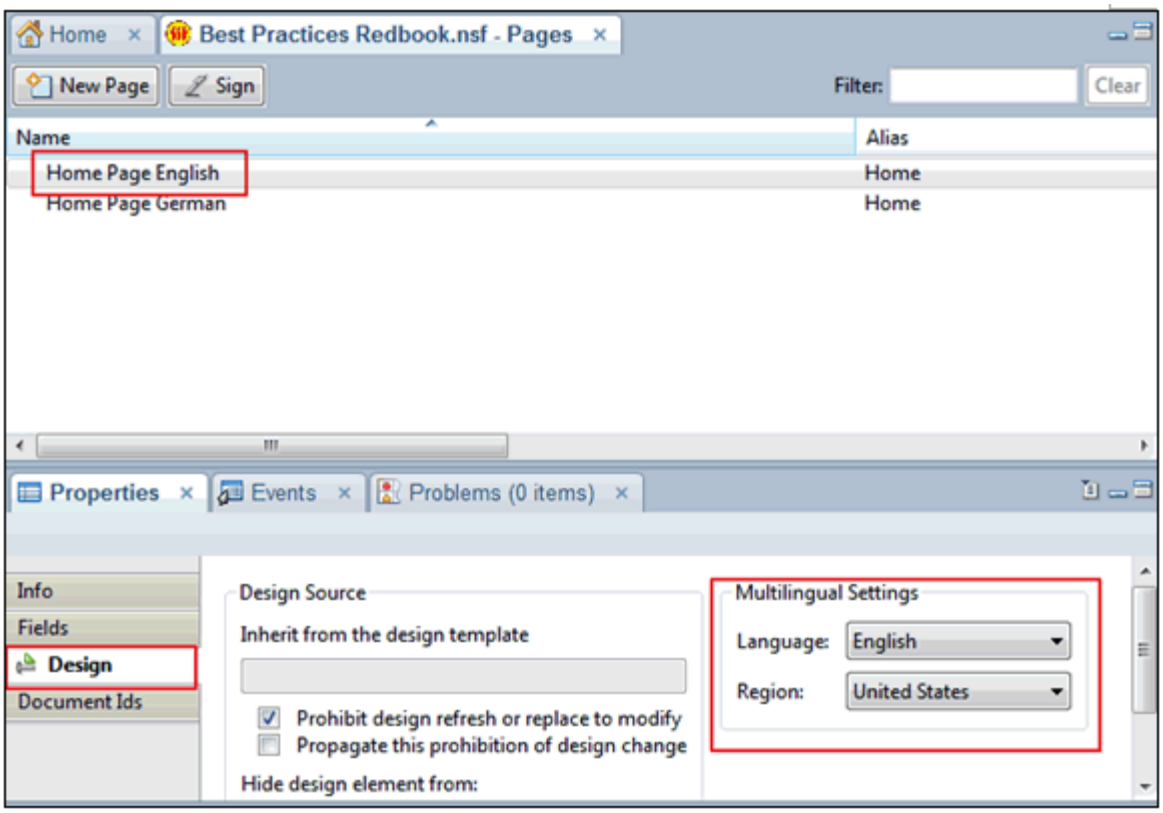


The results of your efforts are shown below:

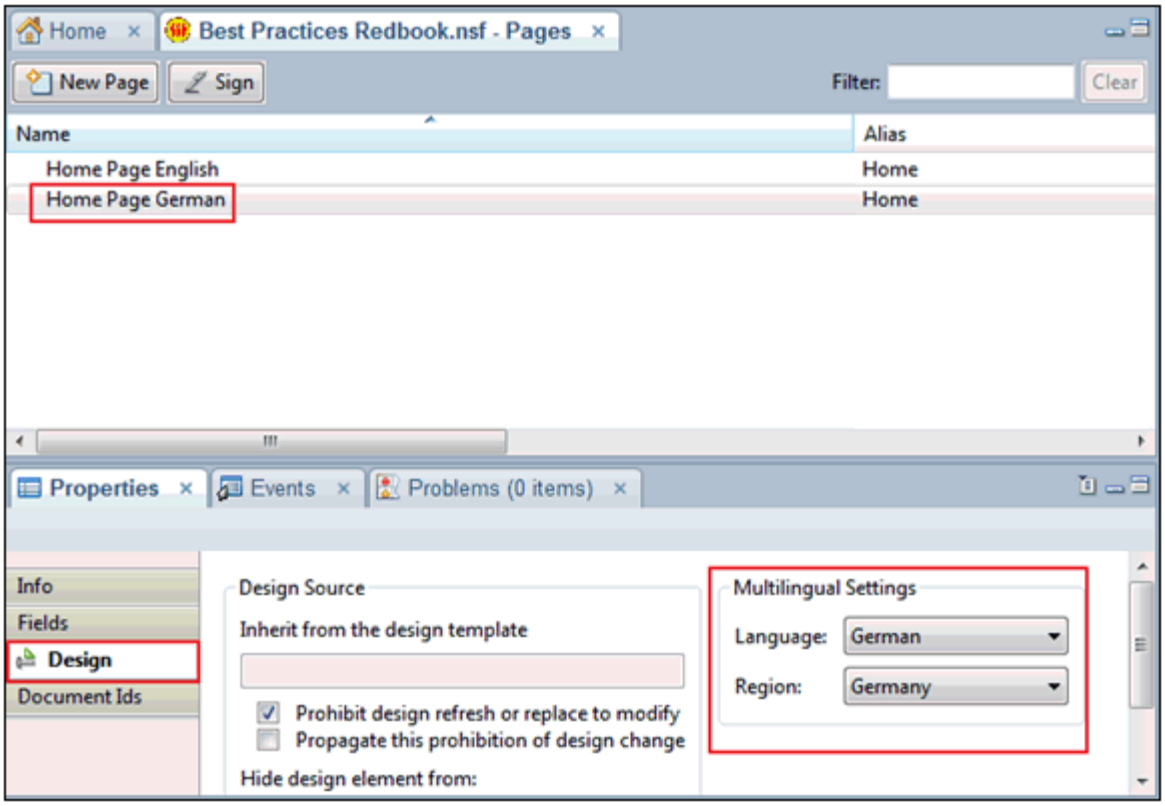


Step 3: Instruct designer to display the relative design element based on the web browser default language setting.
Now that you have two pages named Home, you must tell Domino which page to display based on the default language preferences setting in the user's web browser.

You accomplish this by selecting the page in the Domino Designer, and modifying the multilingual design properties at the bottom of the screen. For example, the Home Page English page would be set as follows:

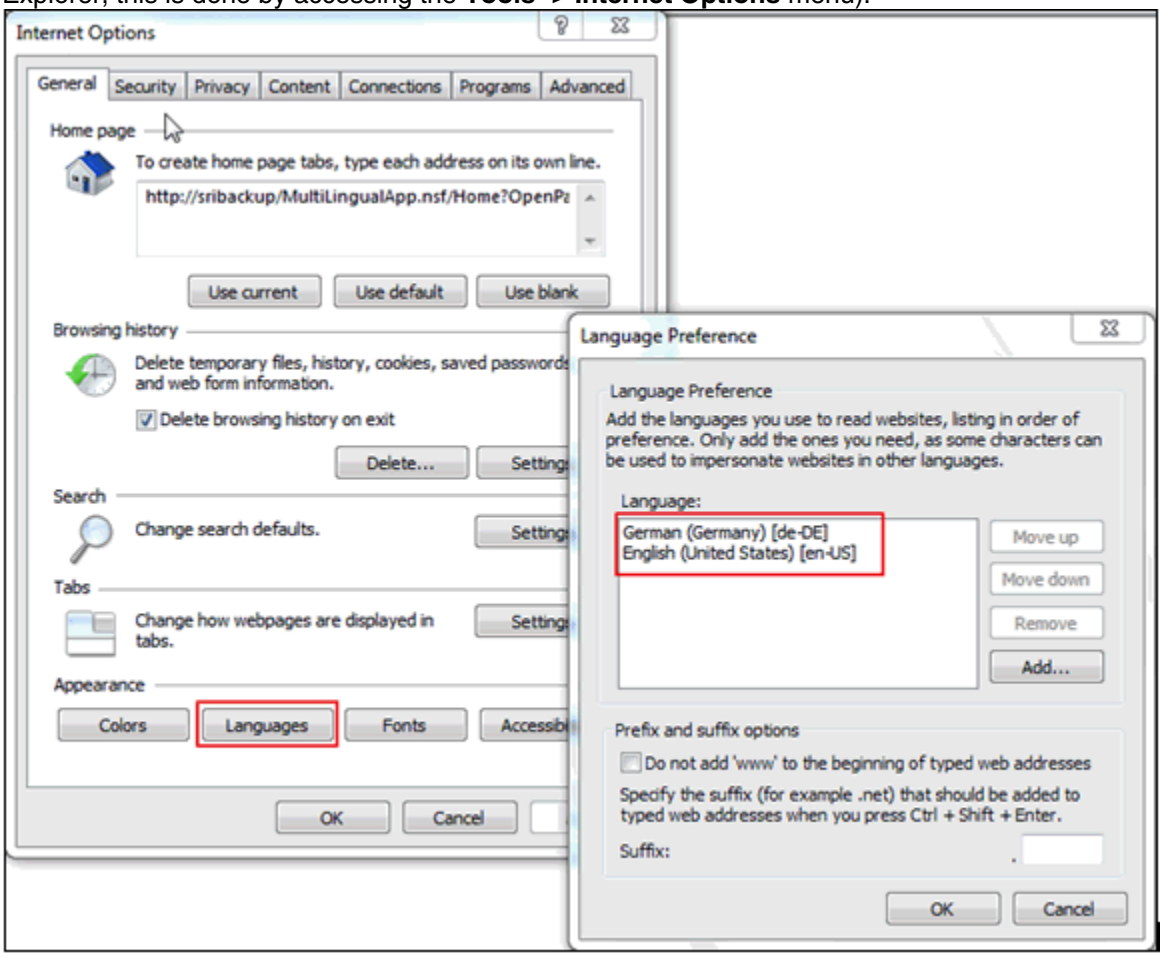


And for German:



Important note: Based on the above, Domino must have some way of knowing whether you want to see the English or German version of the Home page when you access the application from a browser.

This is accomplished using the Browser Language Preference settings (for Internet Explorer, this is done by accessing the **Tools -> Internet Options** menu):



The above browser configuration is set to show the German language version of the page (if available), then the English language version.

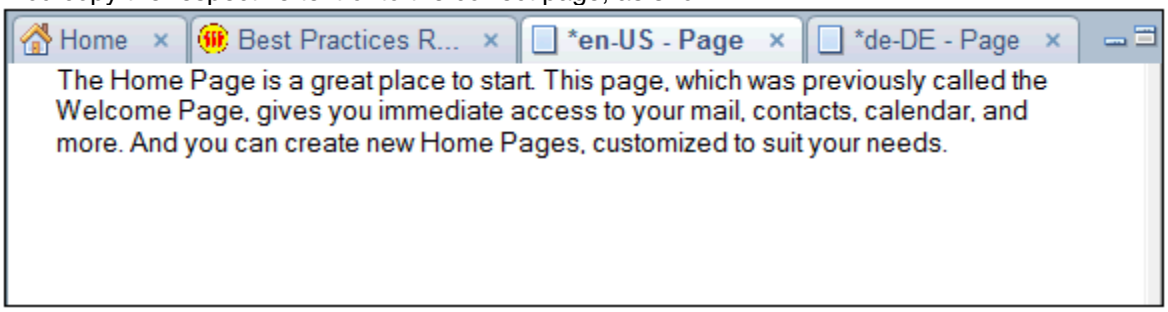
Step 4: Populate the design element with the correct language elements.
Now that you have the web page called Home, one designated for English, and another designated for German, you can now populate each page with its respective text, graphics, media, and so on.

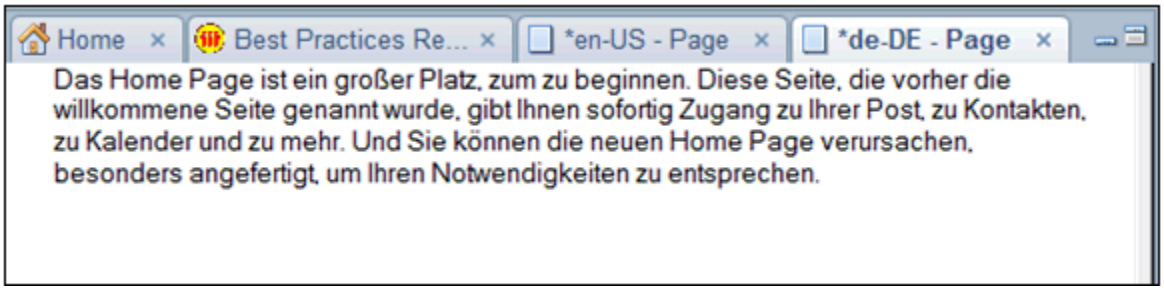
Consider the following text excerpt from the Domino Designer help file:
"The Home Page is a great place to start. This page, which was previously called the Welcome Page, gives you immediate access to your mail, contacts, calendar, and more. And you can create new Home Pages, customized to suit your needs."

To make this available to German users, you must translate the text into the German language, resulting in:

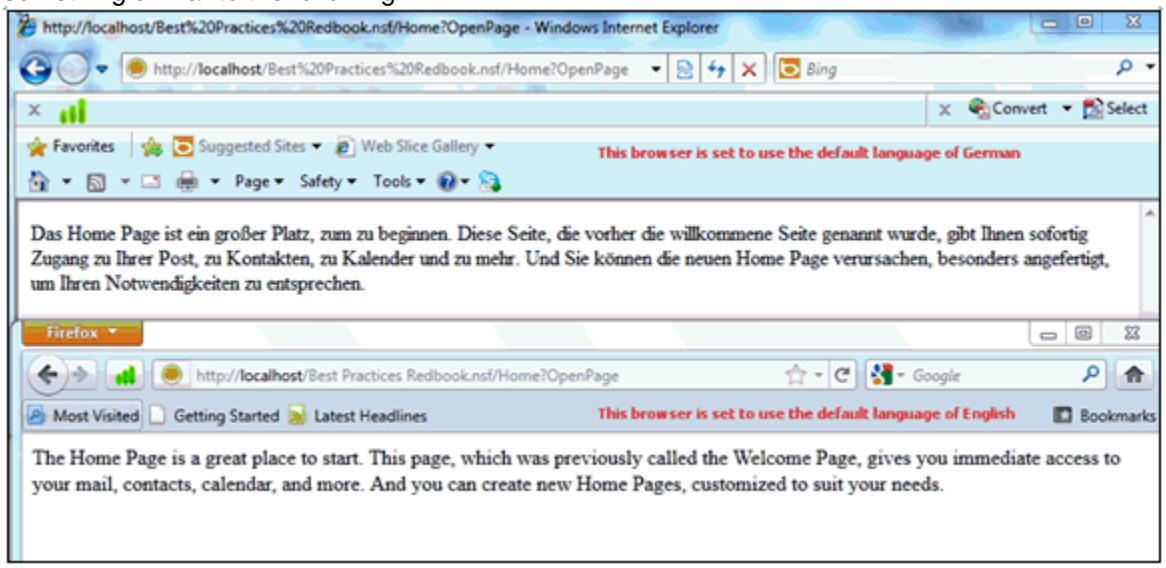
"Das Home Page ist ein großer Platz, zum zu beginnen. Diese Seite, die vorher die willkommene Seite genannt wurde, gibt Ihnen sofortig Zugang zu Ihrer Post, zu Kontakten, zu Kalender und zu mehr. Und Sie können die neuen Home Page verursachen, besonders angefertigt, um Ihren Notwendigkeiten zu entsprechen."

You copy the respective text onto the correct page, as shown:


Home
Best Practices R...
*en-US - Page
*de-DE - Page
The Home Page is a great place to start. This page, which was previously called the Welcome Page, gives you immediate access to your mail, contacts, calendar, and more. And you can create new Home Pages, customized to suit your needs.


Home
Best Practices Re...
*en-US - Page
*de-DE - Page
Das Home Page ist ein großer Platz, zum zu beginnen. Diese Seite, die vorher die willkommene Seite genannt wurde, gibt Ihnen sofortig Zugang zu Ihrer Post, zu Kontakten, zu Kalender und zu mehr. Und Sie können die neuen Home Page verursachen, besonders angefertigt, um Ihren Notwendigkeiten zu entsprechen.

If you preview the Home page in browsers set to different default languages, you see something similar to the following:


http://localhost/Best%20Practices%20Redbook.nsf/Home?OpenPage - Windows Internet Explorer
This browser is set to use the default language of German
Das Home Page ist ein großer Platz, zum zu beginnen. Diese Seite, die vorher die willkommene Seite genannt wurde, gibt Ihnen sofortig Zugang zu Ihrer Post, zu Kontakten, zu Kalender und zu mehr. Und Sie können die neuen Home Page verursachen, besonders angefertigt, um Ihren Notwendigkeiten zu entsprechen.
Firefox
This browser is set to use the default language of English
The Home Page is a great place to start. This page, which was previously called the Welcome Page, gives you immediate access to your mail, contacts, calendar, and more. And you can create new Home Pages, customized to suit your needs.

You have accomplished the desired goal – to create a design element-based web application using Domino.

Best practices

When using design element-based multilingual web applications in Domino, consider the following:

- Use design elements for content that is not stored in documents.
- Use design element internationalization when the content changes infrequently.
- When using design elements, note that the web page that a user will see is based on their default browser language preferences.

3.5.4 Localization using XPages

As with traditional Domino applications, XPages provides a standards-based method for deploying multilingual XPages-based applications. Unlike traditional design element duplication, XPages allows you to develop a single source application to be deployed in multiple languages.

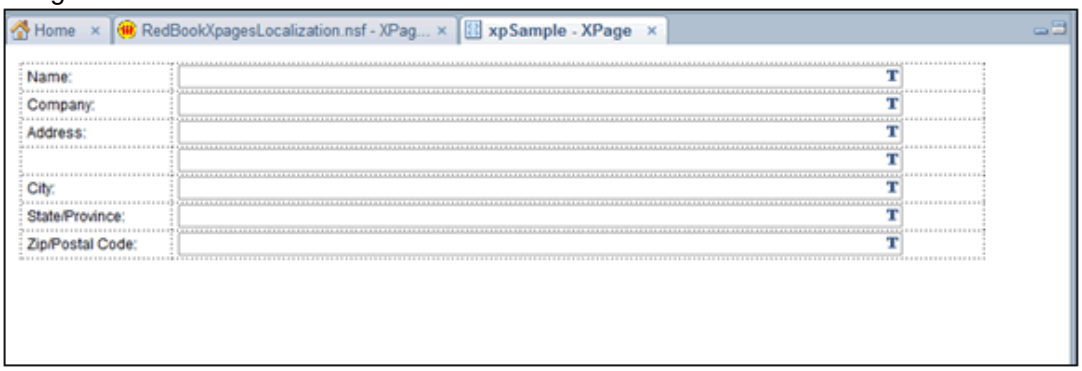
In this section, we show the process for XPages localization and some best practices for implementing multilingual XPages applications.

3.5.5 Localization in-depth

To localize an XPages application, the high-level steps are:

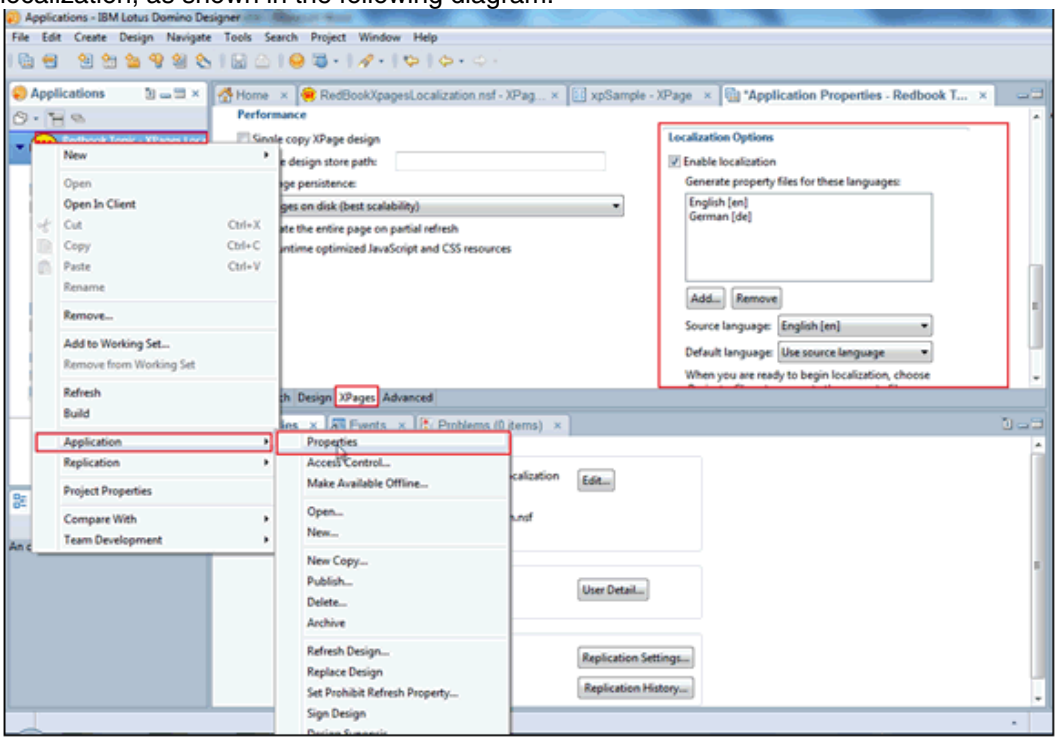
1. **Localization setup:** You must know the native (initial) language that the XPages application will be developed in, as well as configure the application to support multiple locales.
2. **Localization translation:** Once your XPages application is developed, you have to export your native language interpretation and forward them to a professional translator for translation into the additional languages that your application will support.
3. **Localization implementation:** Once you receive your translated information, you import the results into your XPages application.

Here we discuss these high-level steps in detail by creating an example application using the Domino Designer. You also can use Lotus Notes to create an application. Once complete, open the new application in Domino Designer and create the following sample XPage:



Use these steps to apply the localization to this sample application or your application.

Step 1: Localization setup
You must first configure the application to support multiple languages by enabling localization, as shown in the following diagram:



From Domino Designer, perform these steps:

1. Right-click the application. Select **Application -> Properties**.
2. Select the **XPages** tab from the tabs along the bottom of the XPage.
3. Scroll down to enable localization.
4. When done, save your changes.
5. From the Project menu, select the **Clean** option, then, either manually build the project, or automatically build the project.

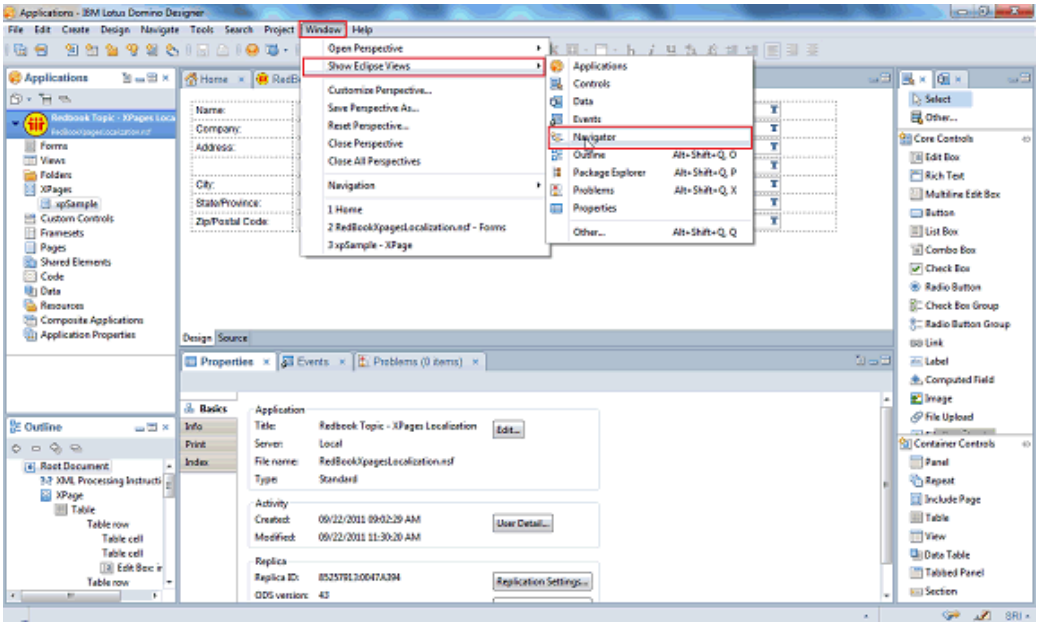
When you enable localization, you must specify the source language (the language that the application will be developed in), as well as any additional languages you wish to support (in this example, German).

When you save and close your application, localization is enabled.

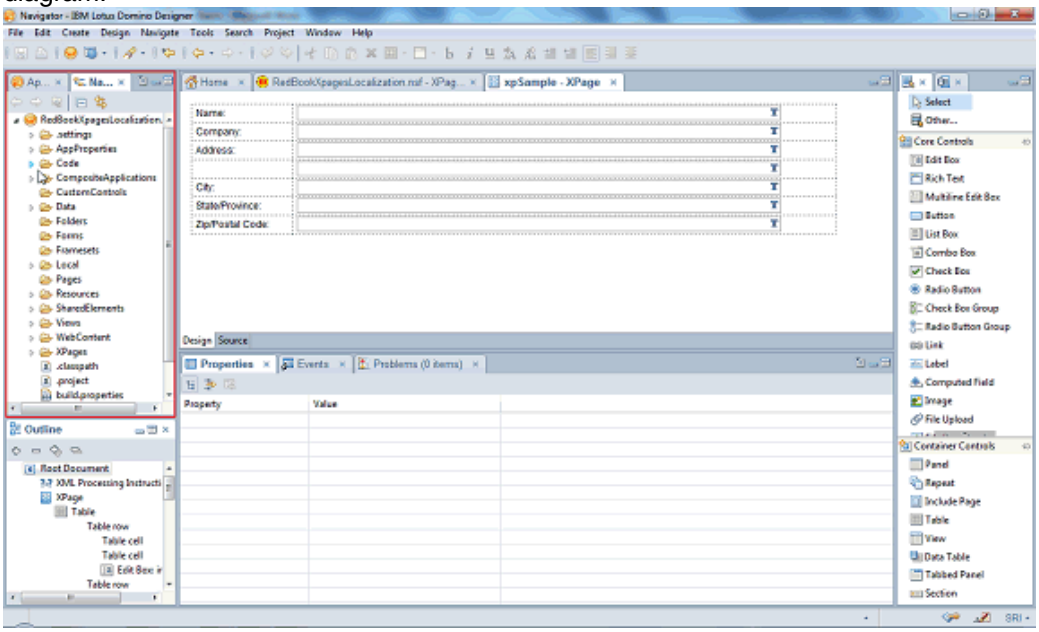
Note: The terminology used in the “Localization Options” of the XPages properties differ slightly for applications developed using Domino Designer 8.5.0.

Step 2: Localization translation
Once the application is developed, you are ready to export the localization information to be translated. This is accomplished as follows:

1. Display the navigator in Domino Designer:
   If the navigator is not present in Domino Designer, you must display it using the From Domino Designer by selecting Window **-> Show Eclipse Views -> Navigator**

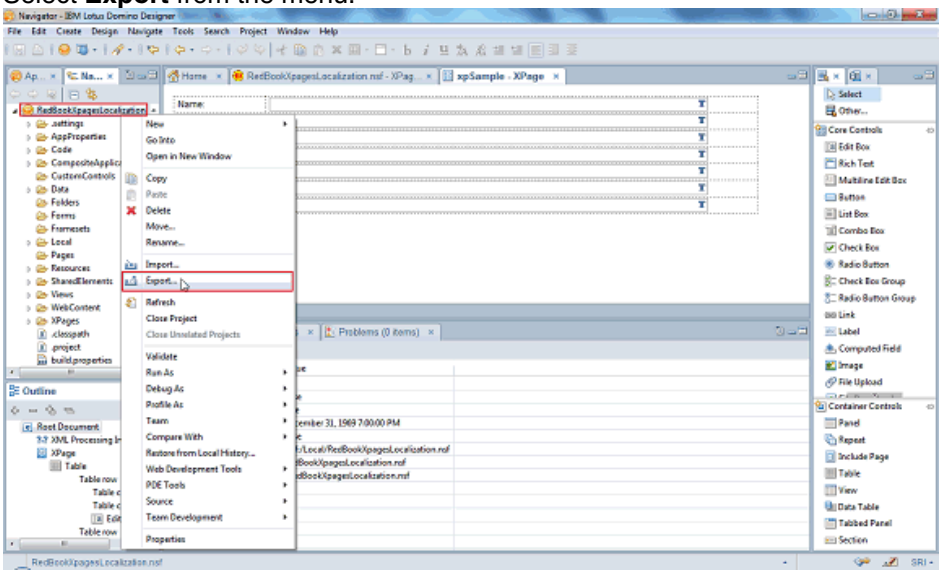


Once complete, your Designer interface should appear similar to the following diagram:

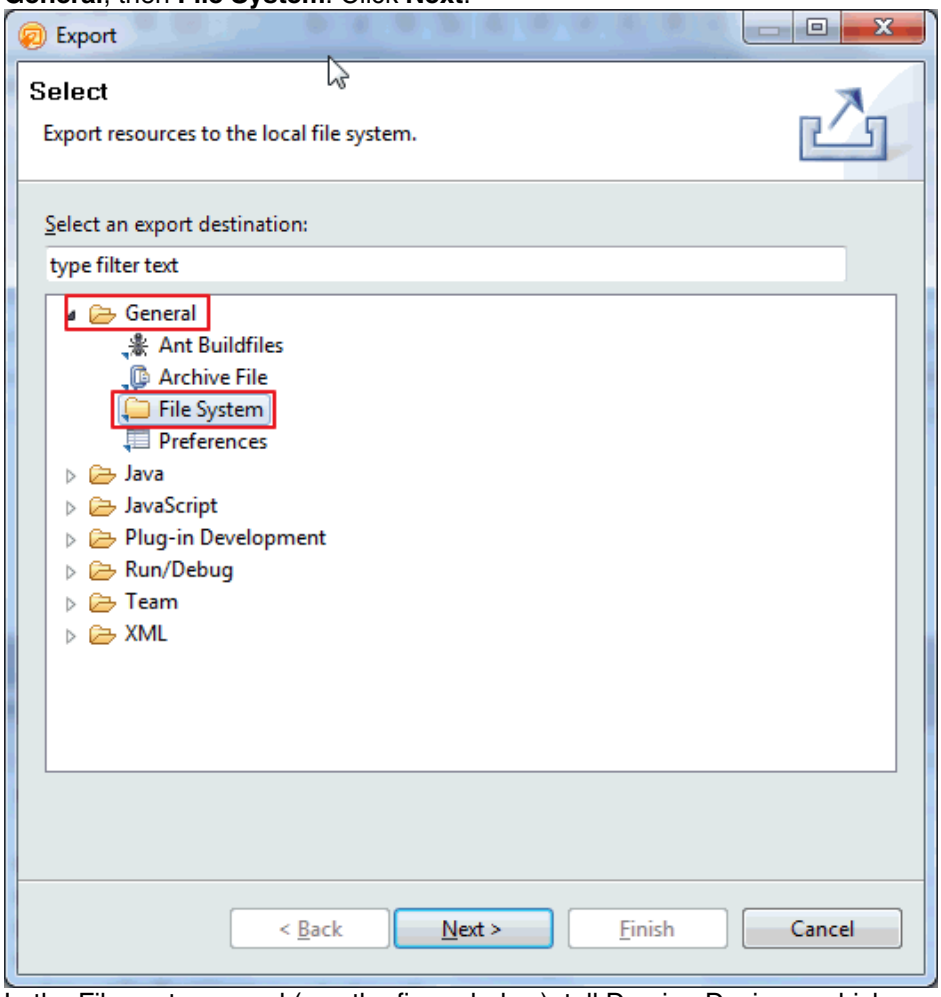


2. Export the information to be translated:
   To physically export the information to be sent for translation, select the following from the navigator:
   a. Right-click the application root.

b. Select **Export** from the menu.

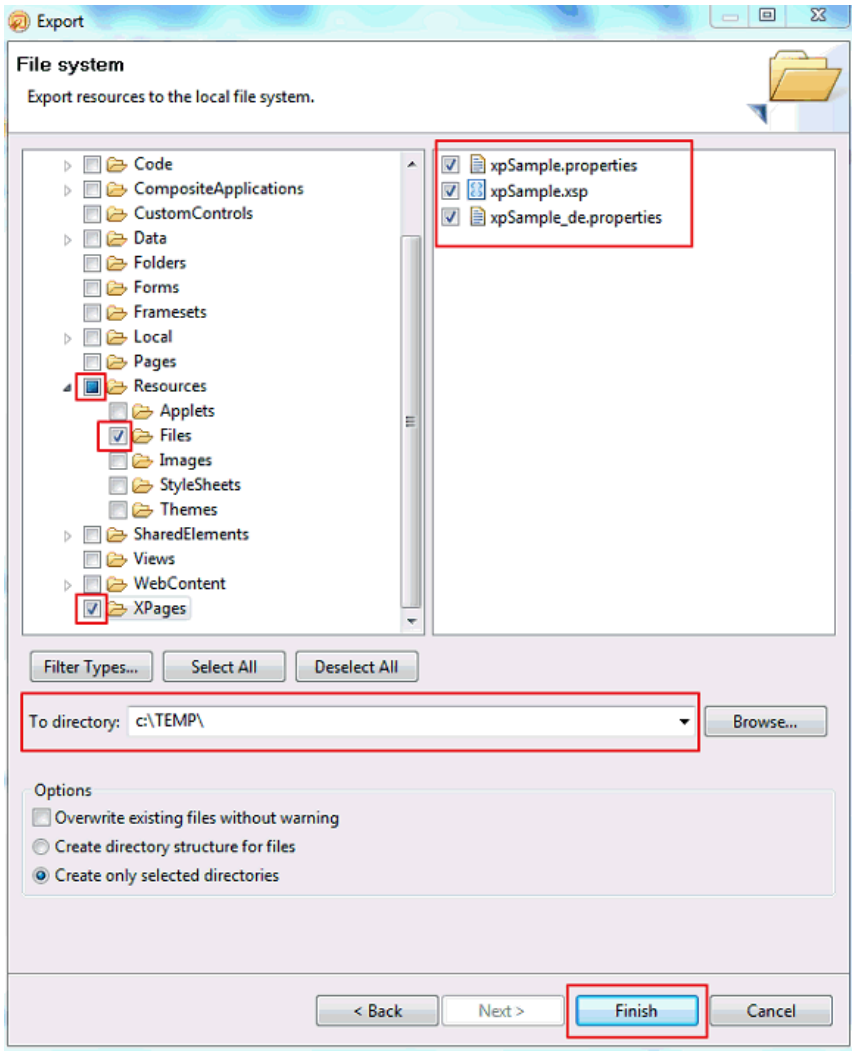


c. The Export dialog box is presented as shown below. In this panel, select **General**, then **File System**. Click **Next**.



d. In the File system panel (see the figure below), tell Domino Designer which components you want to export by:
    - Clearing the checkmark beside the application root.

- Selecting **Resources**, then **Files**.
- Selecting **XPages**.
- Check the .properties files on the right side of the dialog box.
- Specify a path where the files will be exported to



3. When you click **Finish**, the files are placed in the directory you specified in a folder called XPages.

For this example, the following XPages properties file represents the xpSample.xsp file from the dialog box above.

```
#Thu Sep 22 11:57:35 EDT 2011
/xp\:view[1]/xp\:table[1]/xp\:tr[7]/xp\:td[1]/text()=Zip/Postal Code\:
/xp\:view[1]/xp\:table[1]/xp\:tr[1]/xp\:td[1]/text()=Name\:
/xp\:view[1]/xp\:table[1]/xp\:tr[3]/xp\:td[1]/text()=Address\:
/xp\:view[1]/xp\:table[1]/xp\:tr[2]/xp\:td[1]/text()=Company\:
/xp\:view[1]/xp\:table[1]/xp\:tr[6]/xp\:td[1]/text()=State/Province\:
/xp\:view[1]/xp\:table[1]/xp\:tr[5]/xp\:td[1]/text()=City\:
```

Note that this is a simple text file. This simplifies the translation process because the person performing the translation does not require access to Lotus Notes, nor to your

environment.

Also note that a reference to each piece of static text on your sample form is presented in a standard format. This “tagged” file is used by Domino for XPage language translation, and is what you should give to the person performing the translation, who will:

- Open the xpSample_de.properties file. This file will represent the German language version of the properties file.
- Change the text to the right of the = equal sign from its native English to the native German language.
- Save and close the properties file.
- Return it to you for importing into your XPages application.

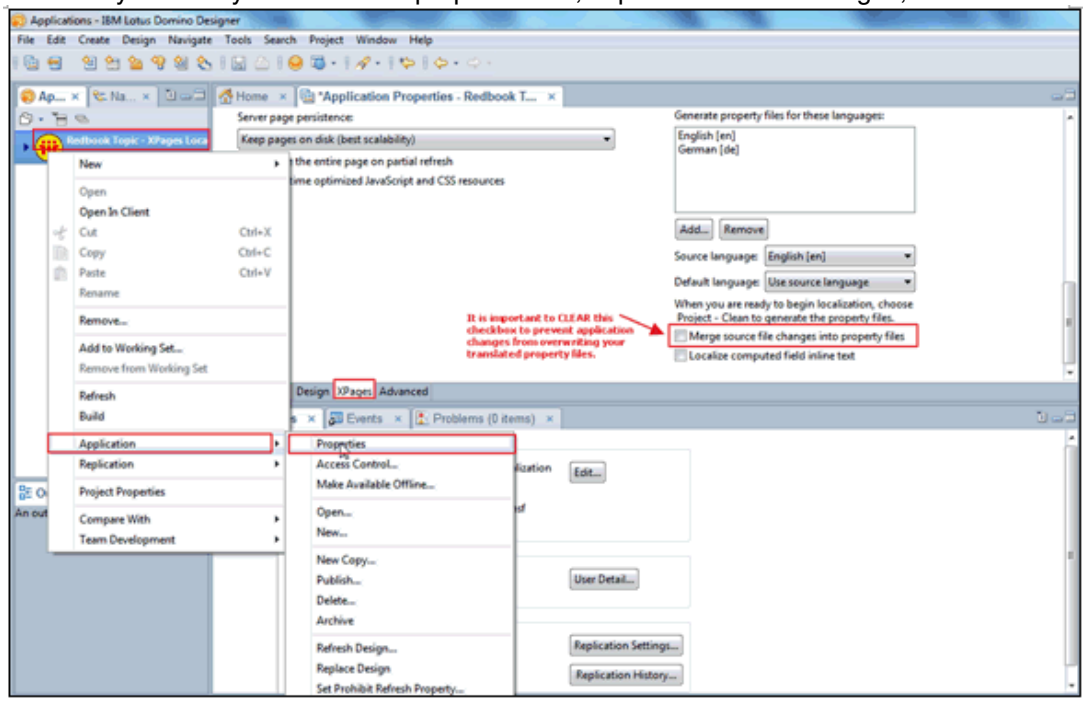
For your sample application, the translated properties file (xpSample_ge.properties) should appear similar to the following:

```
#Thu Sep 22 11:57:35 EDT 2011
/xp\:view[1]/xp\:table[1]/xp\:tr[7]/xp\:td[1]/text()=[de| Postcode\: ]
/xp\:view[1]/xp\:table[1]/xp\:tr[1]/xp\:td[1]/text()=[de| Name\: ]
/xp\:view[1]/xp\:table[1]/xp\:tr[2]/xp\:td[1]/text()=[de| Firma\: ]
/xp\:view[1]/xp\:table[1]/xp\:tr[3]/xp\:td[1]/text()=[de| Adresse\: ]
/xp\:view[1]/xp\:table[1]/xp\:tr[5]/xp\:td[1]/text()=[de| Stadt\: ]
/xp\:view[1]/xp\:table[1]/xp\:tr[6]/xp\:td[1]/text()=[de| Zustand/Provinz\: ]
```

The translator saves this file, returning it to you for integration into your XPages application.

Step 3: Localization implementation

Now that you have your translated properties file, import it back into XPages, as follows:



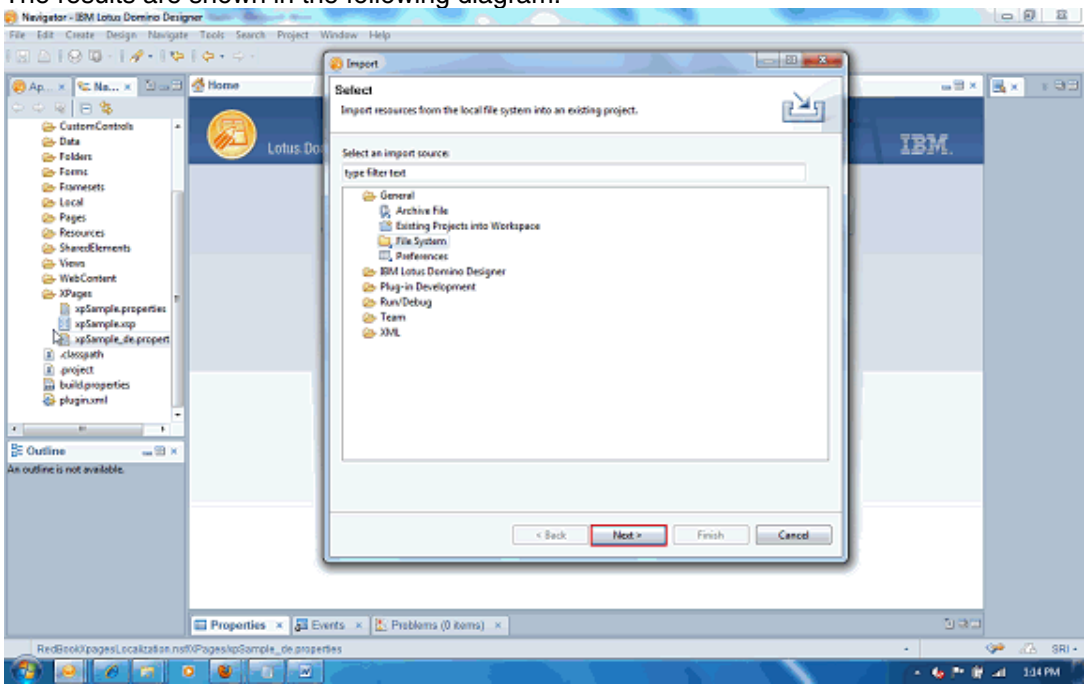
Before you import your translated property files, prepare the XPages application *not* to automatically overwrite the translated property files. This prevents changes to an application from overwriting your translated property files until you are ready to begin the export-translate-import process again. Perform the following steps to accomplish this:

1. From Domino Designer, right-click the application.
2. Choose **Application -> Properties.**
3. Select the **XPages** tab at the bottom of the XPage.
4. Clear the check box “Merge source file changes into property files”.
5. Save your changes.
6. From the Project menu, select **Clean...**

To import the translated property files, perform the following functions in Domino Designer:

1. Using the application explorer, expand the application.
2. Expand XPages.
3. Right-click the properties file you want to import.
4. Select **Import**.
5. From the dialog box, expand General.
6. Select **File System**.

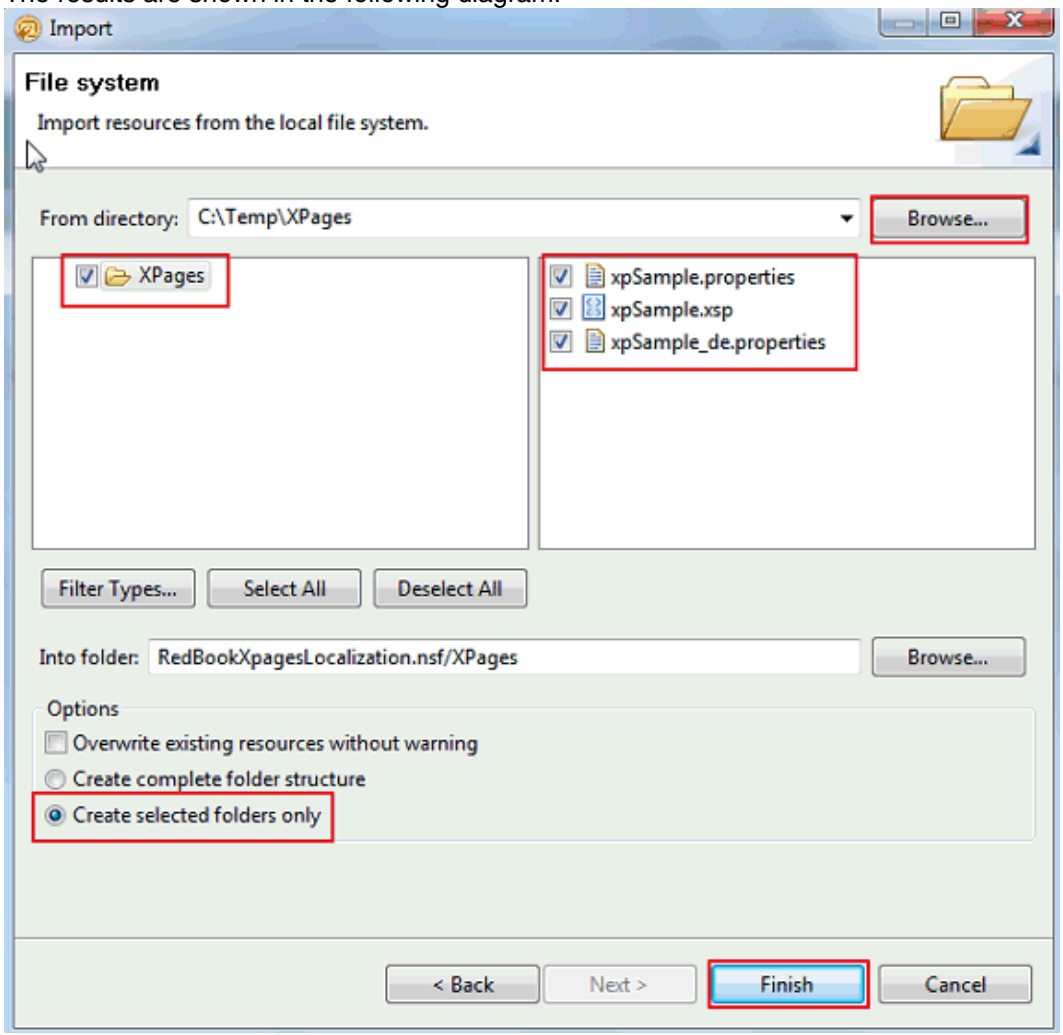
The results are shown in the following diagram:



After selecting **Next**, use the **Browse** button to locate the translated property files on your computer. You must also instruct the import wizard to:

- Import the relevant translated property files.
- Create the folders within the application to hold these files.
- When importing is complete, use the Clean option from the Project menu to assure proper integration of the imported data.

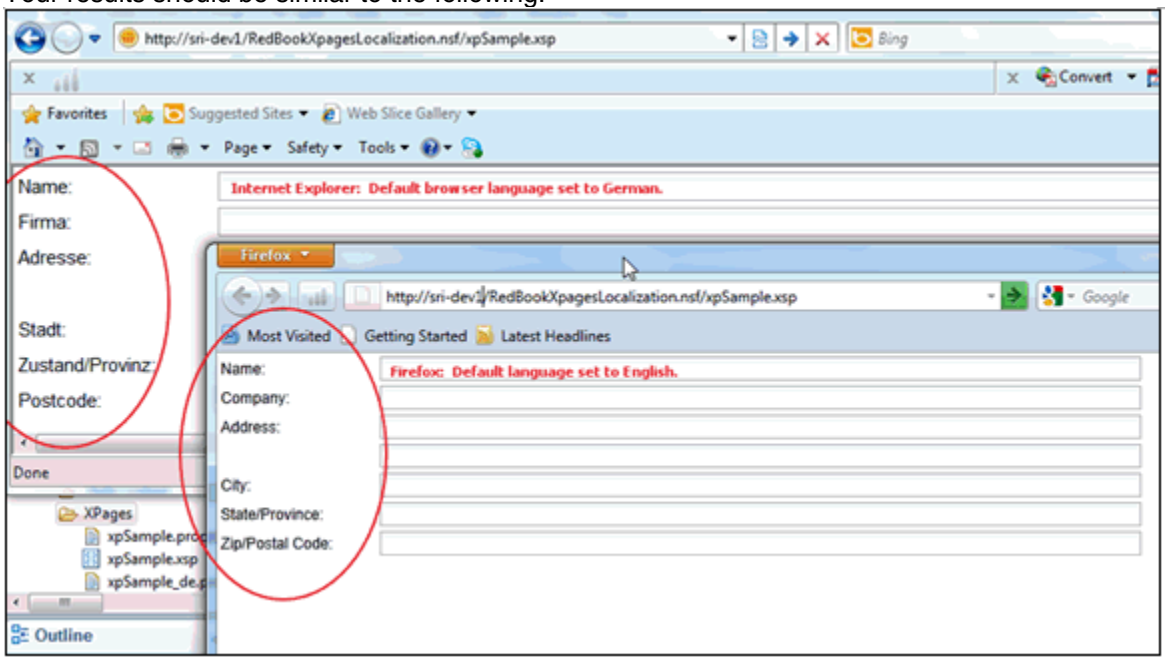
The results are shown in the following diagram:



When you select **Finish** and confirm the operation, Domino Designer integrates the translated language files into your application. You now have a complete, multilingual application that functions as follows:

- You access your application through a web browser.
- The Domino server detects the default language of the browser you are using.
- If the application has the relevant localized properties files, the application is displayed in that language.
- If the application does not have the relevant localized properties files, the application is displayed in the default language.

Your results should be similar to the following:



Best practices

When developing localized applications, consider the following:

- If you are using traditional Domino applications based on the document model, use Domino Global Workbench.
- If you are localizing traditional Domino applications based on the design element model, use design element localization.
- For new and modern Domino applications utilizing XPages, use the XPages localization method. The benefits you receive in XPages localization over other methods include:
    - XPages localization is easier to implement than Domino Global Workbench.
    - Xpages localization is more flexible than design element localization, for you only have one XPage to modify.
    - XPage localization allows translation to be done easily using a text file that is exported from, and imported into, your XPages application. This eliminates the need for Notes or Domino Designer during the translation process.
    - XPages localization is more flexible than other methods, especially when adding additional languages to an existing application.
- When using XPages localization, you should follow these guidelines as a best practice:
    - Make a backup of all properties files – before *and* after the translation.
    - Clean the application after every export and import to assure that XPages properly integrates your localization changes.
    - Clear the checkbox "**Merge source file changes into property files**" on the application properties to prevent simple changes from adversely impacting your translated files.

3.6. Team development considerations

While Domino provides an excellent Rapid Application Development environment that allows a single developer to create some great applications, there are projects that require a team's effort. The most challenging part of working in a team is to communicate what changes are being made so you do not delete another team member's work. As with any code development, you should have a place for the team to see who is working on which design elements and what the changes will be.

Plan your changes to group the changes together into one release. This helps make the testing easier. For each release, you have to coordinate which design changes can go forward and which have to be pushed to the next release.

3.6.1 Work locally

Unless you have a really fast development server, it is better to code locally. There is always a chance that someone else's code might crash the development server and you can lose your work. Replicate often so you have a backup and you get the changes from others. For testing the XPage, use the development server, it provides a better test environment then viewing locally.

3.6.2 Turn on design locking

Replication conflicts between design elements are generally handled be choosing one "winner" and discarding the duplicates. If the discarded design element contains three hours of another team member's work, they may not be happy about this. By enabling the design locking feature of Domino Designer, only one developer will be able to work on a given design element at any given time.

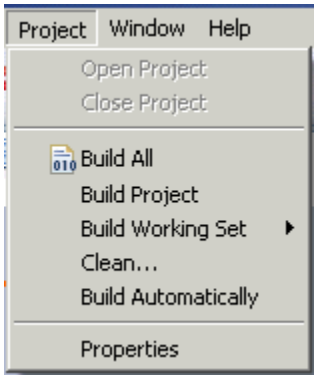
If developers are working in local replicas, this is especially important, since they normally will not get the message about save conflict unless the other developer's change has already replicated to their local replica. Design locking does work in local replicas -- the client will contact the administration server for permission to edit the design element. If the user can't reach the administration server, they do have the option to do a provisional checkout and edit the design element anyway; but make sure they know this is a Bad Idea unless they're sure nobody else is working on that design element.

3.6.3 Turn off automatic project build

By default, project building in XPages applications, is performed automatically when you save changes. If you need more control over when builds occur, you can disable automatic building function and manually invoke the build function. The automatic build is disabled sometime when the building should wait until a large set of changes is finished. The disadvantage of manual building is that tasks generated to indicate build errors quickly become out of date until you build again. In addition, it is important to remember to manually build the project before running those applications that rely on the build output, for example, before running your Java program.

You can turn on and off this option on Workbench using the action menu **Project -> Build Automatically**, which is a toggle menu of on and off. If set to on, whenever a

resource is modified, another incremental build occurs.



Alternatively, you can follow these steps to turn on or of the Build Automatically option:

1. Open the **File -> Preference -> Workspace** page.
2. Select the **Build Automatically** .
3. Click **OK** to close the Preferences page.

While the Build Automatically option makes the build process easy, that you do not have to remember to build to see your changes, it will build all code in the NSF, including changes others have made. As you make changes to an XPage or custom control, you can do a Build Project on each design element. Turning off the build automatically option allows you to shorten the build process and make it faster if you intend to just build the design elements you need to test.

3.6.4 Use same version of Domino Designer for all team members

When you build an XPage, the build process recompiles anything that was complied with an older version or fix pack that has controls that have been updated in the newer version of Designer. This is a real problem if you have Build Automatically enabled since it will build design elements it thinks needs it. The older version XPage or Custom Control will now have your name and the current date and time assigned to the design element. It will look like that you have changed the design element while you had not actually touched it. When you replicate back to the server, you could overwrite changes made by the developer using the older version of the Designer.

3.6.5 Assigned code changes

If possible, assign changes by the design elements that will be modified so you do not have two developers trying to update the same design element. The code libraries give the best chance of over ridding someone else changes. Code libraries allow you to centralize your LotusScript, Java, and JavaScript code. As part of the communications of which design elements are to be change, do not forget to include the code libraries.

3.6.6 Use a third party tool to control the changes

There are tools available to help with controlling the development process. These tools run from the simple code documentation through to design element check-in and out. The version 8.5.3 of Domino Designer supports using a source version control. This is discussed in 3.7 Code control.

Visit OpenNTF.org to see what open source tools are available to help you manage your development.

3.7. Source control

There is no source code control system built into Domino Designer in Eclipse (DDE). However, beginning with version 8.5.3, you can integrate DDE with third party source control products for Eclipse developers.

3.7.1 Background

All the design elements that make up a Domino application are physically stored in a single NTF or NSF file. Eclipse projects, however, traditionally use distinct files stored on the file system in one or more subdirectories. DDE bridged this gap by converting a single NTF or NSF file into a virtual file system so the different design elements can properly show. Third party source control products rely on physical files instead of virtual files. DDE 8.5.3 introduces a way to export a virtual file system into physical files and then automatically synchronize between the two, allowing Eclipse source control products to work on the physical files.

3.7.2 Exporting an Application

To export an application, perform these steps:

1. Open the application in DDE 8.5.3, then go to **File -> Team Development -> Set Up Source Control for this Application**. (This menu option is also available when right-clicking on a database in DDE 8.5.3). The following dialog box appears:

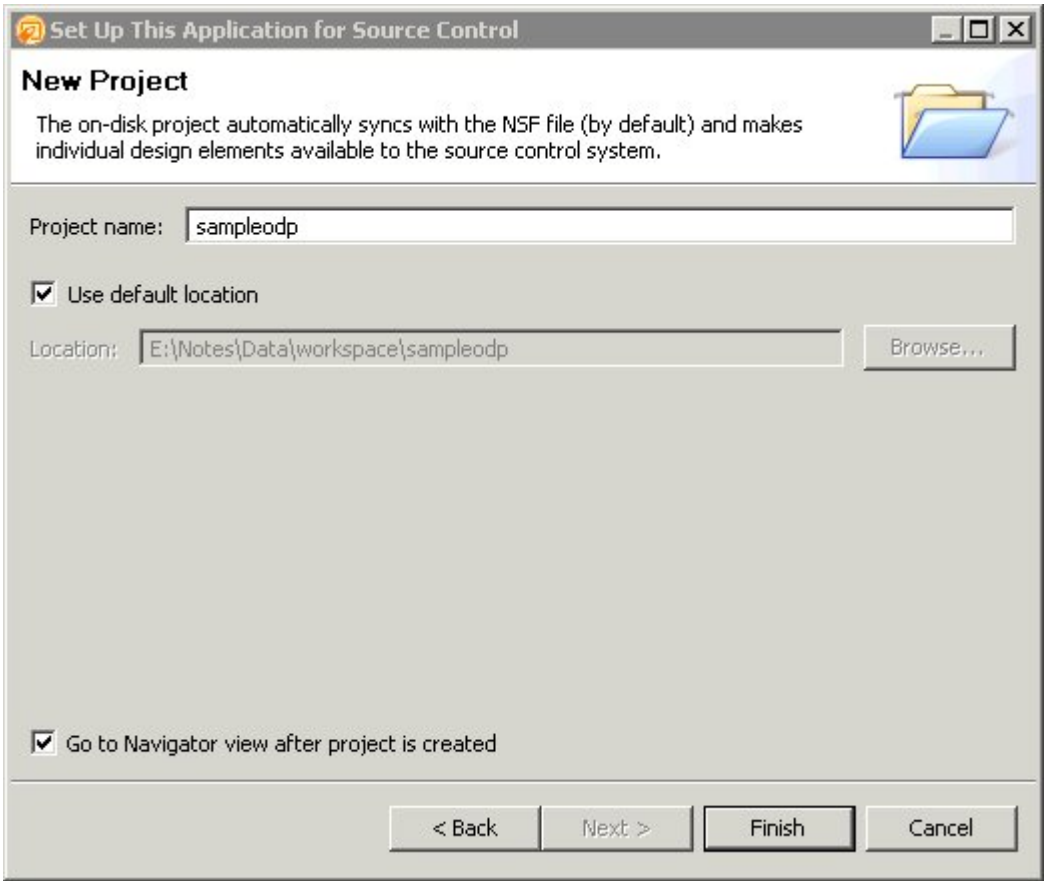
Set Up This Application for Source Control
Get Started with Source Control
To use source control with a Notes application, you must create an on-disk project. The on-disk project is a physical copy of the application (NSF).
1. Create an on-disk project. Domino Designer uses a virtual file system to represent a Notes application as a project in Eclipse. To make the application available to source control, a physical version of the virtual project must be created on the disk. This physical on-disk project also gives the source control system access to individual design elements.
The NSF and on-disk project are automatically kept in sync by default
The on-disk project syncs with the source control repository
2. Use an Eclipse view, such as the Navigator or Package Explorer, to see the on-disk project and perform source control operations on it.
Click Next to create the on-disk project.
Skip this Helper panel in the future
< Back
Next >
Finish
Cancel

Note: if the menu option is not there on your 8.5.3 install, make sure the database is open in Domino Designer. You might also need to switch to a different Perspective to make sure the menu option appears.

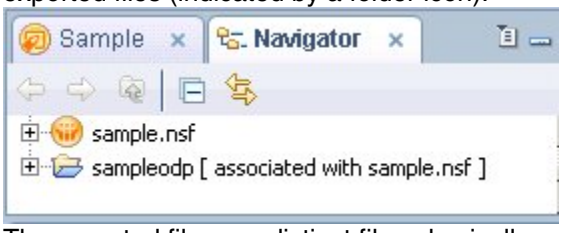
This first screen of the wizard is an introduction and can be skipped in future setups by checking the box at the bottom of the dialog box. Click **Next**.

2. You are prompted to enter a project name. Although this name can be anything you want, it makes the most sense to use something similar to the file name of your application. For example, if your template name is “sample.ntf”, then a project name of “sampleodp” (where “odp” stands for “on disk project”) might be appropriate. The project name will be the base subdirectory name of your project. It can be stored in

the default location or in another location on disk.


Set Up This Application for Source Control
New Project
The on-disk project automatically syncs with the NSF file (by default) and makes individual design elements available to the source control system.
Project name:
sampleodp
Use default location
Location:
E:\Notes\Data\workspace\sampleodp
Browse...
Go to Navigator view after project is created
< Back
Next >
Finish
Cancel

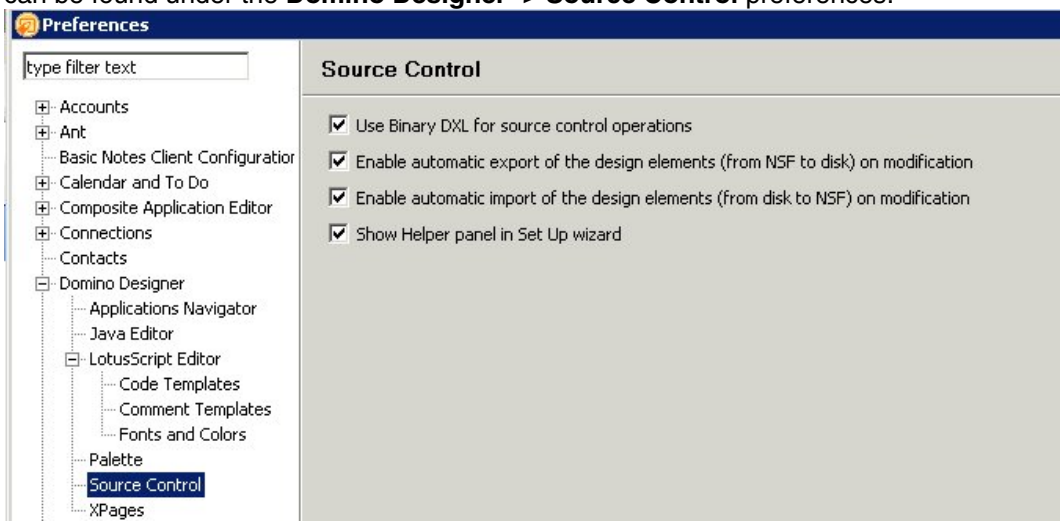
3. After clicking **Finish**, the project is exported to disk. If you selected the option to open the navigator view after the export completed, DDE opens the Navigator view. In the Navigator view, there is the original NTF or NSF (indicated by a Domino icon) and the exported files (indicated by a folder icon).



The exported files are distinct files physically on disk (you can view these files in Windows Explorer). By default, from this point forward, any changes to the NTF/NSF in DDE are also written to the files on disk, and any changes to the files on disk are brought into DDE.

3.7.3 Domino Designer preferences

Domino Designer preferences control the features of Source Control. These preferences can be found under the **Domino Designer -> Source Control** preferences.



Use binary DXL for source control operations

If you have previously worked with text-based Domino XML (DXL) in the past, you might know that there is not currently 100% fidelity between the Domino design element and the text DXL representation of the design element. Source Control uses a binary DXL representation to achieve fidelity between the NTF/NSF and the files on disk. However, this also means that some design elements are not very “readable” on disk. If you have to be able to read through the files on disk and are satisfied that your application can fully use text-based DXL, you can disable this setting. Most developers need this setting enabled all the time.

Enable automatic export of the design elements (from NSF to disk) on modification

With this setting enabled, any change made in the NTF and NSF is automatically synchronized to disk. If this setting is disabled, changes made in the NTF or NSF in DDE are not written to the file system representation (they are only written to the NTF or NSF).

Enable automatic import of the design elements (from disk to NSF) on modification

Similarly, this setting controls whether changes to the file system representation are automatically synchronized with the NTF or NSF in DDE. If the project is not being automatically synchronized, it can be manually synchronized at any point. Go to **File ->Team Development -> Sync with On-Disk Project** and the synchronization will be manually preformed.

Show Helper panel in Set Up wizard

This setting determines whether the first screen of the wizard appears or not. If you decided to skip the first page of the wizard and want to bring it back, enable this setting in preferences.

3.7.4 Third party source control

DDE's implementation of source control was designed so that it does not rely on a source control plug-in for Eclipse from a specific company. Any source control plug-in for Eclipse should work. If you are unfamiliar with source control products in Eclipse, there is a

DeveloperWorks article describing how to use Subversion (SVN) with Eclipse. .

Note that any source control product you use with DDE 8.5.3 must be compatible with Eclipse version 3.4.2, which is a back-level version of Eclipse. This means that sometimes you must download a back-level version of the third party product. You should check the system requirements before purchasing any source control product.

3.7.5 Removing Association

An NTF or NSF that has been previously exported through Source Control can have the connection removed. Go to **File-> Team Development -> Remove Source Control Association**. After confirming the action, the association is removed. This will *not* remove the files from disk, but only remove the association between the NTF or NSF and the files on disk.

3.7.6 Additional Information

Here is additional information concerning the Source Control feature in DDE 8.5.3:

Multiple file system files

If you are viewing the files on the file system, you will notice that some design elements (such as Forms) have one file on the file system and other design elements (such as XPages) have two files on the file system. For an XPage design element, one file is the XML source of the XPage, and the other file is the properties of the XPage (for example, flags such as “do not allow design refresh/replace to modify”). Some design elements require two files on the file system, and others can be contained in one file.

Changes not synchronized

There are certain changes that are not considered significant enough to result in a synchronize to or from the file system. For example, simply opening a design element and pressing CTRL+S without making any changes might not result in the file system representation being updated, even though the time stamp on the design element in the NTF or NSF has been updated. This is expected behavior in DDE 8.5.3. Changes that are significant should result in the files being synchronized. Similarly, if an insignificant

change is made to the files on disk, that change might not be synchronized back into the NTF or NSF.

Chapter 4. Coding techniques

When developing in any language/platform, there are many paths to get to the same destination. Some paths are more efficient than others. While there are few "this is always the case" rules, there are many guidelines that can assist when deciding which coding technique to apply in a certain situation. This section goes through various situations you will encounter in your Domino development, and attempts to provide you with the best practices needed to make your applications shine and perform well.

4.1. Code documentation

Documentation is often overlooked as part of the design process, but should not be. It is important to have developer documentation, user (functional) documentation, and administrator documentation. Documentation can be stored anywhere that makes sense for your application. Some examples of documentation storage:

- In the **Help -> About This Application** and **Help -> Using This Application** documents
- In a separate Notes application (which can be anything from a discussion database to a wiki template to a customized design)
- In files that are stored as File Resources in the application

Regardless of the location of the documentation storage, it should be easily accessible by the target audience. For example, users will make use of the About This Application and Using This Application documents, therefore, if user documentation is stored outside the application, there should be links to the documentation stored in the About and Using documents.

4.1.1 Source code documentation

For developer documentation, one easy thing you can do is to use comments in your code. If you write @-formula code, use the REM statement to include comments. The formula below includes two comments:

```
REM { Get the current role for this user from the profile document for the
user, then filter the view to only show the documents under the specified
category };
CurrentRole := @GetProfileField("profUserSettings"; "CurrentRole";
@UserName);
@SetViewInfo([SetViewFilter]; CurrentRole; "FilterCategories"; 1);
REM { To clear filtering -- @SetViewInfo([SetViewFilter]; ""; 0; 1) };
```

4.1.2 LotusScript comments

If you write LotusScript code, you can add comments in multiple ways:

- A single quote on a line indicates that the rest of the line is a comment
- The REM statement as the first word on a line indicates that line is a comment

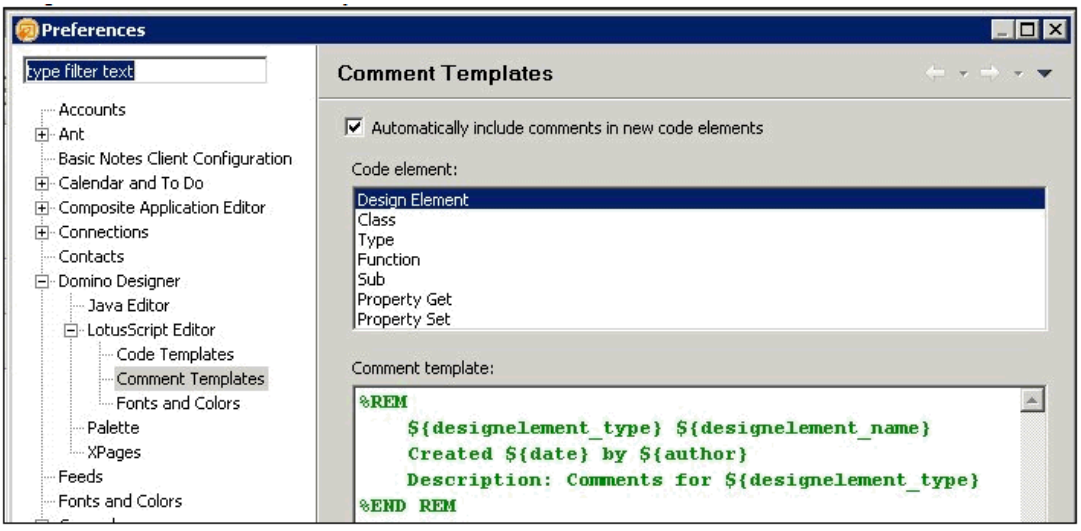
- Lines between a %REM to %END REM block are all comment lines

The following code block shows a block comment (%REM to %END REM), a single line comment (REM), and a comment ending a line (single quote):

```
%REM
     Function AreArraysEqual
     Created Jul 22, 2004 by Joe Developer/New York/ACME
     Description: Take two arrays passed in as input and return Boolean TRUE
          if the two arrays have the exact same elements. The elements can
          appear in any order, but both arrays must have all the same
elements.
%END REM
Function AreArraysEqual(array1 As Variant, array2 As Variant) As Boolean
     REM If the arrays don't have the same number of elements, they aren't
equal
     If Ubound(array1) - Lbound(array1) <> Ubound(array2) - Lbound(array2)
Then
          AreArraysEqual = False
          Exit Function   '  ======== Return to the calling procedure ======
     End If
```

4.1.3 Using comment templates in Domino Designer

If you are using the LotusScript editor in Eclipse, available starting in Domino Designer 8.5.1, by default, comments are automatically be generated within LotusScript code. The editor automatically generates comments for new libraries, classes, types, functions, subs, property gets, and property sets. You can control the content of these comments, disable automatic comment generation, and manually trigger the creation of a comment. To control the comment templates, go to **File -> Preferences** in Domino Designer. Then expand the **Domino Designer** and **LotusScript Editor** categories and click **Comment Templates**.



There is a setting whether comment templates are automatically used or not. If they are being used, you can dynamically generate content through comment template variables. For example, the comment template ${date} can be used to include today's date. You

can fine a full list of code and comment template variables in Domino Designer help. Search for "comment template variables" in Domino Designer helps to find the list of variables.

4.1.4 XPage comments

In your XPage development, your comments depend on where your are writing your code. Inside server side JavaScript (SSJS), use the standard JavaScript comment identifiers. A set of double slashes (//) indicate that the rest of the line is a comment. This can be used on its own line or at the end of a statement. To identify a comment block, that is done with /* at the start of the block, and */ at the end of the block.
Inside the XPage itself, comments are best done using XML comment identifiers within the XPage source (XPage source code is XML). <!-- indicates the start of an XML comment, and --> indicates the end of an XML comment. XML comments can appear anywhere within the XML source code.

Below are code examples from the XML source and the server side JavaScript.

Example from the XML Source of an XPage

The following code block shows an XML comment (<!-- to -->) in the XPage source code:

```
<!-- The below panels are used for tool tips for the buttons on this page -->
<xp:panel id="toolTip_1" dojoType="dijit.Tooltip">
     <xp:this.dojoAttributes>
          <xp:dojoAttribute name="showDelay" value="1500"></xp:dojoAttribute>
          <xp:dojoAttribute name="position" value="below"></xp:dojoAttribute>
          <xp:dojoAttribute name="connectId"
value="#{id:newServerName}"></xp:dojoAttribute>
     </xp:this.dojoAttributes>
     Choose the server to host the application from this drop-down list.
</xp:panel>
```

Example from the server side JavaScript dialog box

The following code block shows both the JavaScript comment block (/* to */) and the comment ending a line (//):

```
     /* There are fields holding the month number and year number. Convert
those
          values to an abbreviated month name. */
try {
     // Build an array of the month numbers
     var sourceArr = "1~2~3~4~5~6~7~8~9~10~11~12".split("~");
     // Build another array of the month names
     var destArr =
"Jan~Feb~Mar~Apr~May~Jun~Jul~Aug~Sep~Oct~Nov~Dec".split("~");
     var title = @If(@IsNewDoc(),
          "New Report",
          "Report for "
               + @Replace(currentDocument.getItemValueString("ReportMonth"),
sourceArr, destArr)
               + " " + currentDocument.getItemValueString("ReportYear")
     );
     return title;
} catch (e) {
     return "";
}
```

4.2. Using library versus coding in agents or events

4.2.1 Comparing XPages to Java agents

Although you really cannot compare an XPage to a Java agent because of their different capabilities, understanding the underlying class loaders gives you a general feel for the performance. In a Java agent, there is a class loader per class each time the agent is called. This means that even if you request the same Java agent multiple times, there is no advantage gained the second time the agent is called. Also, if the agent references multiple libraries, those libraries must be loaded each time. Therefore, a Java agent with multiple libraries does not perform as well as a Java agent with all the code in a single class. But, even the Java agent with all the code in a single class is loaded every time that agent is called.

In XPages, there is a single class loader per instance of the application. It holds the classes in memory until the database times out (a period of time after the last user has referenced the application). This means that multiple users of the same application get a performance gain by sharing the class loader. If you are trying to decide between doing coding in a Java agent or an XPage, you should choose the XPage.

4.2.2 Using JavaScript Script libraries

If there is client-side functionality that you have to perform in your XPage development, you should take advantage of JavaScript libraries. You store your client-side JavaScript code in the library and then reference the functions from your XPage. By developing this way, you take advantage of both client-side and server-side design element caching. The client only has to bring down that design element once and then it can use the cached version on subsequent XPages.

4.2.3 Using server side JavaScript libraries

Whenever possible, you should take advantage of server side JavaScript (SSJS) libraries instead of coding directly inside the XPage. Even though both the XPage and the server side JavaScript are compiled, if you are using a server side JavaScript, you are taking advantage of caching on the server. Multiple people referencing the same XPage do not have to have multiple versions of the code loaded into memory.

4.2.4 Development considerations

Coding in libraries instead of directly in the XPage is also easier for the developer. When viewing the XPage source, it will not be cluttered up with a lot of JavaScript and server side JavaScript code. By isolating the code, your XPage source is cleaner and easier to support. You not only gain performance benefits, but also receive support benefits.

4.2.5 Other considerations

Starting in Domino 8.5.2, there is an external servlet engine used by the Domino server. This servlet engine performs very well. If you can take advantage of creating servlets to be used by Domino, this can help your performance because of how servlets are loaded and maintained in memory.
You can gain performance improvements by taking advantage of web services too. By using a web service, the amount of code you send to the client is reduced and the logic happens on the server.

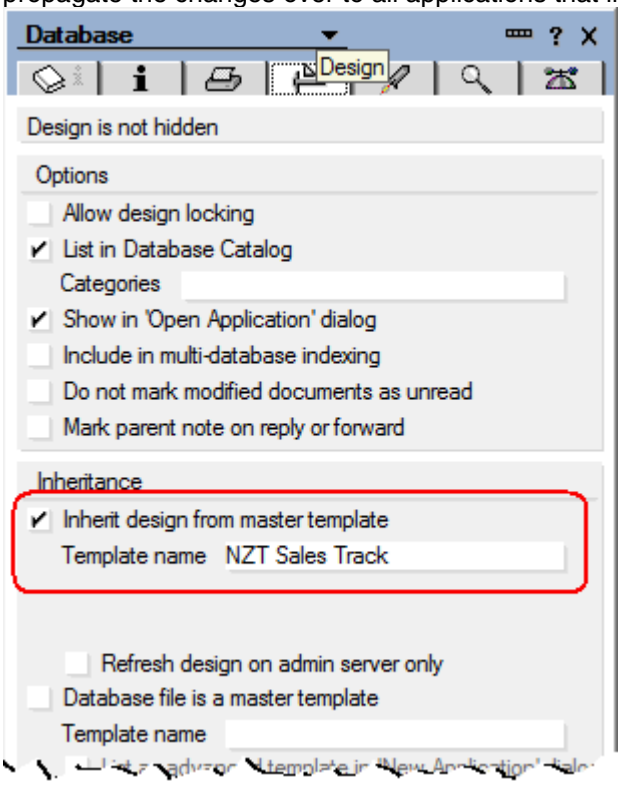
4.3. Using templates

In today's high productivity business environment, all the application development, including the Domino applications, are expected to deliver consistently effective results in an optimum manner. In all application development activities, source code development requires a major labor effort. The Template (.ntf) is associated with Notes and Domino as a mechanism for source code control management processes. It is imperative to manage the source code using templates to achieve cost effective results in a consistent manner. The benefits of template are not just limited to the new application development but also for the enhancement of the existing applications. If your application is identical, or similar to, an existing template available in your Domino application portfolio, the most convenient way to create a new application is to use that template as your starting point. This is a common scenario where the new application requires almost all features similar to an existing template. There could also be some scenarios wherein you need features

to be inherited from a cluster of templates.

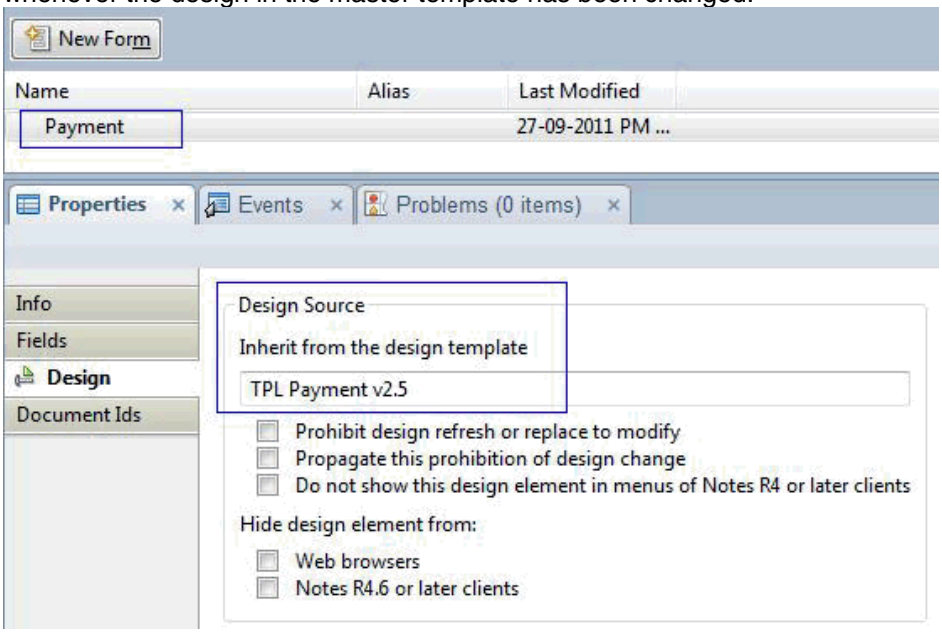
The following are scenarios of using templates:

- Application level inheritance:
  It is not uncommon that the business wants to deploy several applications that all use the same code features. However, certain work-groups, for example, a product quality department, would want to maintain their own set of data to be managed privately. Instead of maintaining separate copies of application code for each department, best practice for the development team is to separate the databases for each department and have the applications that inherit the code or design from a common master template. This approach allows each work-group to maintain their private data but share the same code. Any future improvement in the application features is to be implemented only in the master template. You can configure the Database (see the figure below) so that the design refresh task would automatically propagate the changes over to all applications that inherit this template.



- Design level inheritance
  It is common for a Domino applications in a portfolio to have several common features and also has its own distinct features simultaneously. The best approach in using templates in this scenario is as follows:
  - The application inherits its design through the database level inheritance activated in the database property "Inherit design from master template" for the major functions.
  - The additional features in this application could be inherited from more than one template.
  - Create a new master template and have the code inheritance at the design level that allows certain design elements inherit its code from one template and others from another template (See the figure below). Such design elements need to be copied and pasted from the master template to the application. These design elements would update itself automatically

whenever the design in the master template has been changed.



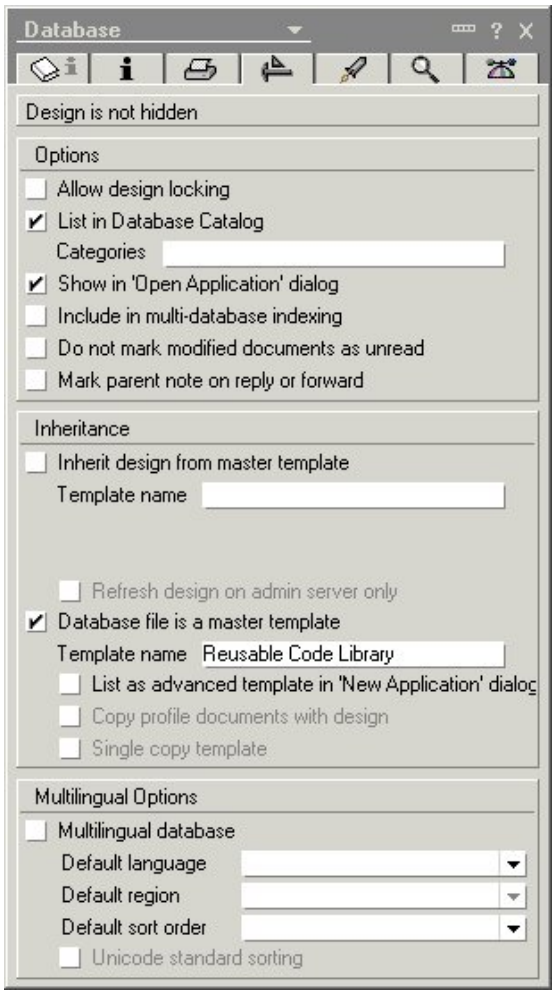
- Template documentation
  Templates are used among work groups and it is important to document each template clearly. It helps the source code version control management process. All templates should incorporate documentation. It is a good practice to document your work especially when you are working with templates that might have many developers working in conjunction to build the finished product. Using the built-in documentation that Domino provides can greatly enhance your template in the long run. Examples of built-in documentation include:
  - About This Database
    This mode of documentation introduces the template to all users. It describes the purpose of the template, who might use it and when, and the recommended access control settings for databases created with the template. The best practice to document about the template in About this Database is to divide it into two logical sections. The first section list down the features and functionality that is built-in to this template. It also list down the version history details (for e.g. Version Name, Number and Date) and the development team responsible for maintenance of the template. The second section has the technical details or embedded a technical specification document. This helps the team members old and new to be aware of the technical implementation of the design.
  - Using This Database
    The Using This Database document gives instructions about the usage of the application and its features. It has to be updated with more focus to the end user of the application. The document list down the steps and procedures to carry out the activities to make use of the features offered by the application. The application information in this section is expected to be the user manual for the end user training on this application.

4.4. Resource templates for design elements that are inherited across applications

One effective way to help jump start any new application development work is to have a repository of generic, well-tested code that performs specific tasks. For example, you might have a function that prompts the user for a file name and return the name of the file. Instead of the developer having to write the code using the NotesUIWorkspace class and do the appropriate testing to see if the user chose a file name or clicked cancel, the function performs the specific task.
Once you have all these functions built up from your own experience, or from reusing code found on the Internet or other resources, you need a location to store this code so it can be quickly retrieved by developers.

4.4.1 Reusable code library

A Notes application can be used to store the reusable design elements. The application can be either an NSF or an NTF, and should be designated as a template.



It's a good idea to use the comment field of the design element to note that it came from the code library, and if it is a text-based element (script library, agent, stylesheet) also put a comment to that effect in the text. This lets later developers know where to go to find the latest version, and also makes it simple to automatically catalog all design elements in all applications whose comment fields show them as coming from your repository, so that you can find out which applications need to be updated when an element in the repository is updated (assuming you have access to all those templates to scan them).

While it is technically possible to allow the design element in the using template, to continue to inherit design changes from a repository template, in practice this is rarely a good idea, since changes to the repository design elements may easily cause applications that use them to fail if the changes are not done extremely carefully. We advise extreme caution in letting changes roll out into production applications; as described elsewhere in this wiki, it's best to retest the application in a copy of the design, before rolling out changes to production. Developers of some applications may need to continue using an older version of something until they have adapted their application to accommodate the change.

For LotusScript libraries, even changes that don't involve adding parameters may cause calling code to need to be recompiled before it will work with the new version of a library.

You may want to create an accompanying registry database where developers who make use of a library element, can record that they are using it. This gives you a mailing list of people to be notified when a master copy of a design element has a new version (and also lets you see who's contributed the most reused code).

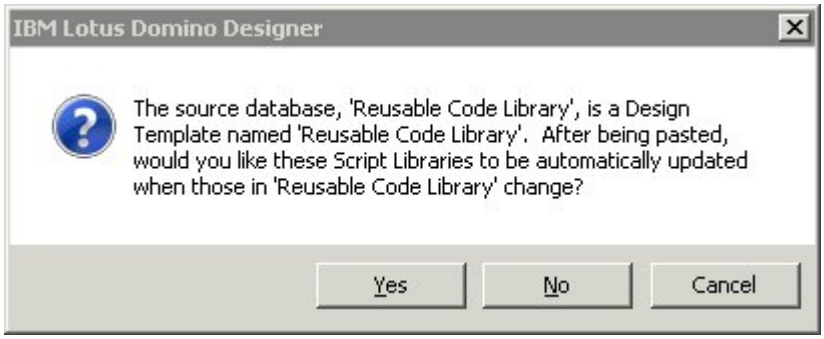
4.4.2 Reusable code library design

In addition to storing the design elements inside the code library application, it would make sense to include a simple form and view in the design of the code library application. The form and view would include documentation of the design element, so developers could easily find the design element they need instead of looking at agents, script libraries, custom controls, and so forth.

4.4.3 Using code library design elements

NOTE: There's some disagreement on the recommendation here. See the above for the reasoning why the following is a bad idea, then you decide what's best.

When the reusable code library has been built, developers simply need to copy and paste the design elements into their application. Choose **Yes** to the prompt about inheriting from the Reusable Code Library template name:


IBM Lotus Domino Designer
The source database, 'Reusable Code Library', is a Design Template named 'Reusable Code Library'. After being pasted, would you like these Script Libraries to be automatically updated when those in 'Reusable Code Library' change?
Yes
No
Cancel

By answering “Yes”, the design element inherits bug fixes and other updates to the original code in the reusable code library template.

4.5. Forms, Views, and XPages coding considerations

Traditionally, Domino applications, simple or complex, use of *forms* and *views* that are the classic design elements for building Domino-based applications. The form and view design elements are part of Lotus applications since IBM Lotus Release 1.0. Over the years, form and view design capabilities are enhanced continuously to keep pace with the changing business landscape and the technological advancement. Continuing with the tradition of enhancing the Lotus Domino platform and strengthening the developer community ability to efficiently build Domino applications with rich features, IBM Lotus Release 8.5 introduces a wrapper design element, XPages. XPages brings flexibility to build Domino applications because of data and presentation are separated. Data isn't stored with the XPage. Instead, the XPage is linked to a data source, such as a Domino form or view. XPages allows more user interface options and greater control of appearance with less effort and greater performance efficient. XPages has many built-in capabilities for building the Web 2.0 rich features in the Domino applications. XPages enable us to develop more performance efficient Domino applications with the use of custom controls.

In this section, we discuss the designing and coding consideration for form, view ,and XPages in Domino application development.

4.5.1 Form design and coding considerations

Using the XPage features is a best approach to develop your Domino application data presentation features with substantially less efforts. It is helpful to have a form to provide a list of fields for binding data so the documents can be opened from a Notes client. You also can use the form to calculate the computed fields with field formulas rather than putting code on the XPages. Refer to 5.3 XPages versus forms for comparison, usage, and best approach about using both design elements to build Domino application.

Consider the following when design and code forms:

- Use tables to align elements on a form. Nested tables give you precise control over how you present content. In addition, you can use tables to create certain text effects, such as having text wrap around a picture.
- Group related information. Use sections for approvals or other special access needs.
- Use collapsible sections to organize a form and make it easier for users to see the information they need. Set the section properties to expand the section in one context and collapse it in another.
- Use the div tags to generate sections in your form to provide better control over the layout formatting of the sections.
- Create subforms that group design elements you could use in multiple forms.
- When designing multiple forms for an application, locate particular fields, especially data such as name, department, current date, and due date, in a consistent place and a consistent order.
- Place hidden fields together at the bottom or top of a form. Assign a different text color to hidden fields.

- Computed fields are evaluated from the top down and left-to-right. Place dependent fields after the fields they depend on. Forgetting this rule is a common source of problems.
- Use centered text at the top of a form only. It tends to get lost when used farther down on the form.
- Use consistent spacing between elements, and avoid crowding the information together.
- Hide elements that users do not need to see when they are editing, reading, or printing. In particular, hide nonessential graphics while printing; alternatively, provide a separate read-only form with suitable data layout as a printer-friendly output option.
- Provide actions and hotspots to let users take action quickly. It is a best practice to make use of HTML markup text to create the action buttons or links powered with JavaScript code to perform basic user interface validation on the form to be submitted.
- Use anchor links to move to a paragraph within a document. Anchor links, like named, anchor in HTML pages, allow you to link to a particular area in a long document. Use anchor links when documents contain a large amount of related information and want readers to find a particular section. For example, an anchor link to go to the "Top" of the form.
- If a form contains many fields that contain summary data, it might have an impact on the performance of views because all the summary data has to be loaded as part of the collection. Even with this in mind, however, you should still try to compute as much as possible within the fields on the form instead of in the view.
- Web browsers interpret the original text font, size, and style. In your design, you can only approximate the look and size of text, which may appear different to a web user.
- Web browsers ignore tabs, indents, inter-line spacing and layout regions. Use tables to align form elements. To add graphic interest, add embedded views and navigators.
- Try to take the advantage of computed-for-display fields for those calculations that do not need to be stored, such as the document created date. This is always available with the function @Created and can be displayed in a computed-for-display field. Though eight bytes may not seem like a significant savings, however, if you stored it for every document in a database with 30,000 documents, it begins to add up very fast.
- Check the fields that are calculated and stored based on a constant-value fields. Use computed-for-display to reduce the total size of a document and thus to improve performance.
- The formulas in fields do not affect server performance because documents are created on workstations. The validation and input translation formulas can have performance impact if a user is running on a very slow machine with little memory.
- Functions that have impact on performance should be used sparingly. For example, the @Db functions, @DbLookup and @DbColumn, can significantly affect both server and workstation performance. When using these functions, it is often a good idea to create specific views just for that specific purpose, and have the views select only the document that are absolutely needed by the lookup. Limiting the number of documents in a lookup view and keeping those views up to date can increase performance of these functions. Also, use the cache (default) wherever possible.
- Minimize the number of summary fields in each document.
- Under Design - Document Information, if the field flag is set to SUMMARY, then that field information is displayable by the view. The view index stores field information for all summary fields, whether displayed in the view or not, in preparation for evaluating the selection formula and the fields displayed by the view. All fields other than rich text fields are summary fields.

- The space required for header information for a blank or empty field is approximately seven bytes. This means that if you have 50 blank fields in 10,000 documents in a database, there is about 3.5 MB of wasted space.
- Use @DeleteField in the input translation formula of editable fields which might remain blank. If the field is called SubjectLine, the formula would be as follows:

```
@If(SubjectLine = ""; @DeleteField; SubjectLine)
```

- In most formulas, Notes treats a nonexistent field the same way it would treat a blank or empty field; therefore, although you do sometimes need to code around these fields, it is no more coding than you would have to do for the cases where these fields were empty or blank.

Determining when to use subforms

To simplify the maintenance of your design, create subforms containing elements such as specific section (approval, banner, and so on), actions, fields, and formulas or scripts that you can share across many forms. You can also group shared fields, which in themselves streamline your application on a subform. You can then make a design change once and automatically update all the forms that use that design element. While subforms simplify form maintenance, loading subforms can slow the opening of documents slightly.

Determine when to use collapsible sections

Use collapsible sections to:

- Make a form shorter and more modular.
- Control user access to parts of a form.
- Provide complex user interface objects that, if rendered directly on the form, would slow down the opening of documents created with the form. Use iFrame to load such section, it helps divide the load of form section-wise on demand when user expands the section for first time.

Performance consideration of forms

A form that calculates formulas quickly opens the documents created with the form quickly. To improve form performance, do the following:

- Limit or eliminate shared fields or subforms on any form that must open quickly.
- Architecturally, shared fields are on a different note and Domino must make extra internal function calls (a NoteOpen) to get the value of the shared field -- one NoteOpen call to get the document, another NoteOpen call to get the form, and third one to get each of the shared fields. However, you should still consider using these design components in the form to gain the benefits of ease in maintenance over the slight delay.
- Limit the number of fields on the form to the minimum required to have a lean, clutter-free form with better response time in opening documents.
- Remove infrequently used items from a form. For example, redesign your application to display infrequently used items in a dialog box.

- Minimize the number of fields on each form because each field is calculated when a document is opened, refreshed, or saved. After your design is complete, run an agent to remove any blank and unused fields.
- Avoid over usage of hide-when formulas on forms. Each formula that must calculate when a form opens slows your application down. Before you use a hide-when formula, try using a computed subform or a hide-when condition such as "Hide when editing" or "Hide when reading." If you have Web-only or Notes-only forms, give them the same name and use the Design Properties box to hide them from the inappropriate audience.
- Make use of computed-for-display fields to calculate the hide-when computation only once and use this field at all the places in the form. This helps reduce the number of computation for the server thereby helps improve the forms performance.

Web form performance considerations

The following table shows tips for improving web form performance:

Tip	Comment
Don't use field names that start with HTTP_	Domino assumes that "HTTP_" indicates an environment variable. The search for a non-existent environment variable slows performance.
Minimize field computations	Remove or reduce the number of computations. If you have two computed fields whose values are supplied by a potentially slow formula such as @DbColumn, use one computed-for-display field to collect the data, and compute the other fields from the computed-for-display field.
Use single-use fields	Forms and documents that use shared fields open more slowly than those with single-use fields.
Keep image file sizes as small as possible	
Simplify formulas	Complex formulas and those that use @DbLookup and @DbColumn may process more slowly on the Web. In Notes, formula processing is shared between the Notes client and Domino server. On the Web, all formulas are processed through the server. Consider using JavaScript within the form's HTML to achieve the same results. JavaScript runs on the browser, so it reduces the server's load.
Avoid form formulas in views and multiple hide-when formulas in forms	To present different information to Web and Notes users, create sets of computed subforms that have the same name and different hide-when conditions. Alternatively, create two versions of a form, one for the Notes client, which is hidden from Web clients, and one for Web clients, which is hidden from Notes clients.

Improving the performance of lookup formulas

@DbLookups and @DbColumns are extremely useful, however, they can be quite a performance drag on a large application. Keep the following in mind when creating applications that use @DbColumn, @DbCommand, and @DbLookup to search for and retrieve information from databases. Use these rules of thumb to minimize the cost of your @Db functions:

- Minimize the number of @Db functions in any document.
  If there is a standard group of fields that are returned through @DbLookup, use one @DbLookup to return a set of values (which contains a list of the contents of all those fields), and then parse that field out in your document. If the look-up is required to fetch multiple field or column values for the same key, use the following approach to minimize the number of @DB functions:
    - Make lookup views with computed column to list the values of multiple fields using a separator:

```
 EmpName + “~” + EmpDept + “~” + EmpDOJ + “~” + EmpLocation + “~” +
@Implode(EmpSkills; “^”)
```

Here the “~” character is the field separator and the “^” character is the multi-value separator of a particular field.

- Use a combination of @DBLookup, @Word, and @Explode formula functions to perform single lookup and split values of separate fields. Using this approach for @DBColumn you have to be cautious to anticipate the amount of look-up data to be fetched in real business scenario. It should not exceed the 64K data limit for formula look-up.

- Have @Db functions refer to small, external databases with minimal design elements.
- Perform look-up on column numbers rather than field names if performance is more important than easy code maintenance. Looking up a view value is faster, however, keep in mind that if the column position changes during a view redesign, you will need to change all formulas that reference the column.
- For documents that are often just read (as opposed to almost always being edited), you should use @IsDocBeingLoaded or @IsDocBeingSaved as part of the formula, or use AJAX capabilities to look up the data on demand with a button clicking or a user triggered action.
- Use the Cache argument in @DB functions whenever possible if the look-up value does not change frequently.
- Perform look-up on hidden views that have a minimum amount of data.
- Avoid redundant look-up in a formula by setting a temporary variable for the first instance of a returned value and then referring to the variable for the rest of the formula. Alternatively, use a computed-for-display field to calculate the computation once and re-use this field reference wherever applicable in the form.

Generating HTML for hidden fields

Enabling the "Generate HTML for all fields" forms property generates HTML information about hidden fields on a form. This allows the document behavior in a web browser to match the document behavior in a Notes client more closely. For example, if you create a form that relies on a hidden field for a calculation, that form may not behave as expected on the web. By generating HTML for the fields, the information is available for Domino to complete the calculation successfully. The HTML generated for the hidden fields is also accessible through JavaScript, so you can change the value or find out the state of a hidden field using a script.

To generate HTML for hidden fields, use these steps:
1. Open the form properties box.
2. Click the **Defaults** tab.
3. In the "On Web Access" section, check **Generate HTML for all fields** and uncheck **Treat document contents as HTML**.
Cautions:

- Checking this option creates larger files on the web, and may degrade application performance.

- Consider security, since information in hidden fields, though not visible in the browser, is visible through the "View Source" menu item on the browser.

Using hidden fields

Most forms, in both Notes and the web, include hidden fields that pass constant values. On the web, the hidden input fields include the tag "type=hidden." If your application will be used from both Notes clients and web browsers, and you want hidden fields to perform consistently in both clients, you must follow these steps:

- Do not put "type=hidden" into the HTML Body Attributes objects of the hidden fields.
- Make sure that you select **Use JavaScript when generating pages** in the database properties.
- Select **Generate HTML for all fields** on the form properties.

Domino will generate HTML hidden input elements for most hidden fields. This allows JavaScripts to have access to all the Notes fields in the form, and makes debugging easier by allowing all the fields to be included in the HTML source code for the document. The fields of type text, number and time and their range/list equivalents are supported. The composite field type is not supported. Several other specialized field types are also not supported for security reasons, for example, No HTML INPUT statement is generated for password fields, "$Updatedby", "$Htmlhead" , or the section fields associated with controlled-access sections. Item values that contain NULLs are not output either because there is no way to express them in HTML. User public keys fall into this category of objects.

If you have selected "Use JavaScript when generating pages" in the properties box of a database and you want to use hidden fields on a form in that database, you must also select "Generate HTML for all fields". It is not necessary to set each field "HTML Body Attributes" property to "type=hidden". These settings helps in debugging and they must be on if using JavaScript to refer to a hidden field.
Note: Enabling JavaScript and HTML generation for all fields should not be used if you are hiding fields for security reasons. A web browser view source function will display the field contents.

Using a file upload control instead of adding attachments in a rich text field

The rich text fields in applications intended for the web cannot be used for attaching files to the document as they can in Notes. If a user attempts to attach a file, the file will simply be dropped when the form is transmitted from the browser to the server. If you want users to be able to make attachments, create a file upload control in the form. If the application is used from both the Notes clients and the web browsers, add an additional rich text field to be used for attachments from Notes clients, set the hide/when properties of the upload control and the rich text field to display the proper client.

Passing hidden field values to the browser

To pass on information about a web document, such as its current state, the Notes document ID, and other variable information, include the hidden field values in the form used to create the document. These values are returned to the server when the web user

submits the document. Create a hidden value by adding an editable text field and hiding it with an HTML field event. The document must be in the edit mode for this technique to work. The steps are as follows:

1. Create an editable text field on the form that will be used for creating documents.
2. In the HTML Attributes field event, enter **TYPE=HIDDEN**.
3. Select **Hide paragraph when document is opened for reading** in the Hide tab in the Field Properties box.
4. Write an agent that generates the field value and run the agent from a WebQueryOpen form event.
5. Write an agent that saves the field value and run the agent from a WebQuerySave form event.

Note that this is not a security feature. Users can see the field value when they view the document HTML source code.

You should write agents instead of using a computed field because:

- If neither of the "On Web Access" property options for the form are selected, the hidden values are not returned to the server by the browser.
- If "On Web Access: Treat document contents as HTML" is selected, Domino performs no computation on the form. The computed field is ignored when the form is passed to the browser as HTML.
- If "On Web Access: Generate HTML for all fields" is selected, Domino converts computed fields to HTML text when it sends a document to the browser for display. Only editable fields (not HTML text) are returned to the server when a user submits a document, so the hidden values are not returned to the server.

The reason that we do not use the use "Hide paragraph from Web browsers" property is that when the field property "Hide paragraph from Web browsers" is selected, Domino skips the field when it converts the document to HTML and the field is unknown to the browser. To make a field known to the browser without letting the web users see it, you must use HTML to hide the field.

Making field names and values in HTML difficult to find

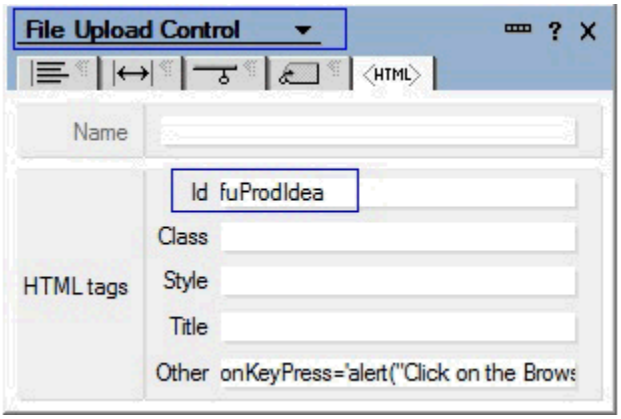
The only way to make hidden fields accessible to JavaScript is to set the form property to "Generate HTML for all fields". However, this feature will expose field names and item values that were not exposed before. The only way to keep the hidden field names and values hidden is to not enable JavaScript and HTML generation. The browser's "View Source" command displays the HTML code, including field names and values, that make hacking easier. You cannot turn this off, however, you can make it more trouble for the hacker to capture the code by keeping the pages out of the browser cache, which normally holds pages and forms even after exiting the browser. You can write the <META> tags within the page's <HEAD> tag to set the page to expire on a date that has already passed, and use the Pragma parameter to tell the browser not to cache the file. Put this into the "HTML Head Content" object for the form:

```
 "<META HTTP-EQUIV= \"expires\"content = \"Wednesday, 27-Dec-95 05:29:10
GMT\"><META HTTP-EQUIV=\"Pragma\" CONTENT=\"no_cache\">”
```

Attachments links in sections of a web form

In Notes client interface, you can place a rich text field is placed in various sections and categorize the attachments to a specific section. However, in the Domino application for web browser, the files are uploaded using file upload control, the files are attached to the document and not associated with any specific rich text field. The association of the attachment with the section is thereby lost. Considering a business requirement to display the links of attachments specific to a section from that the document is uploaded. The following approach helps you overcome this limitation and provide the attachment links in a specific section:

- Assign an ID to the File Upload Control (see the figure below). This ID is for the JavaScript code to identify the file attachment selected for upload through the respective file upload control. Store the names of file attachments uploaded through this file upload control in a dedicated field that indicates the association of the attachment with the section.



- Use the attachment names in the dedicated field to generate the attachment links in the respective sections.
- Check to ensure the valid attachment names are recorded in the dedicated attachment names field:

```
onKeyPress='alert("Click on the Browse button to add a file.");return
false'
```

- Capture the Ctrl key and mouse right-click event to avoid junk data entry in the file upload control. The check for duplicate attachments name should also be put in place to restrict upload of such attachment, to avoid loss of the association with section due to the automatic change in attachment names by Domino for duplicate attachments.

Understanding the META tags

You can use the <META> tags in many ways to enhance your site, ranging from identifying the creator of a page to what HTML specifications the page follows to preventing browsers from caching pages. HTTP-EQUIV and NAME are two commonly used attributes in <META> tag:

- HTTP-EQUIV attribute
  The HTTP-EQUIV attribute of the <META> tag is the equivalent of an HTTP header. To understand what a header is, you need to know a little about what actually goes on when you use your web browser to request a document from a web server. When you click on a link for a page, the web server receives your browser request through HTTP. Once the web server has made sure that the page you have requested is indeed there, it generates an HTTP response. The initial data in that response is called the "HTTP header block." The header tells the web browser information that might be useful for displaying this particular document.

  Similar to the normal headers, the HTTP-EQUIV values usually control or direct the actions of web browsers, and are used to further refine the information that is provided by the actual headers. The HTTP-EQUIV values are designed to affect the web browser in the same manner as normal headers. Certain web servers may translate META HTTP-EQUIV values into actual HTTP headers automatically so that the user's web browser would simply see them as normal headers. Some web servers, such as Apache and CERN httpd, use a separate text file that contains meta-data. A few web server-generated headers, such as "Date", might not be overwritten by the <META> tags, but most will work just fine with a standard web server.
- NAME attribute
  The <META> tags with a NAME attribute are used for the meta-data types that do not correspond to the normal HTTP headers. Note that certain search engine agents (worms and robots) interpret tags that contain the keyword attribute whether they are declared as "name" or "http-equiv".

Using <META> tags

The tags should always be placed in the HTMLHead event of your Domino pages and forms to ensure that they appear before the <BODY> tag. This is very important with framed pages and is frequently missed in the individual framed pages.

One of the examples in implementing <META> tags in a Domino application is to prevent web page caching in users browser so the users will always see the updated pages. By using the <META> tags, you can tell the browser not to cache files as well as when to request a newer version of the page. The following are attributes that can be used to prevent browser caching problem for dynamic data presentation in Domino applications:

- Expires
  This tells the browser the date and time when the document will be considered "expired." An illegal date such as "0" is interpreted by the browser as "immediately." Dates must be in the RFC850 format (GMT format), for example:

```
<META HTTP-EQUIV="expires" CONTENT="Fri, 13 Feb 1990 05:12:32 GMT">
```

- Pragma
  This is another way to control browser caching. To use this tag, the value must be "no-cache". The following example use two tags together to keep your content current.

```
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
```

Note that many users have reported that Microsoft Internet Explorer refuses the <META> tag instructions but caches the files anyway.

- Refresh
  This value specifies the time in seconds before the web browser reloads the document automatically. Alternatively, it can specify a different URL for the browser to load.

```
<META HTTP-EQUIV="Refresh" CONTENT="0;URL=http://www.lotus.com">
```

Be sure to remember to place the quotation marks around the entire value of the CONTENT, otherwise, the page will not reload at all.

- Set-Cookie
  There are a variety of ways to implement cookies into your application, this is one method of setting a cookie in the user's web browser. If you use an expiration date, the cookie is considered permanent and will be saved to disk (until it expires), otherwise, it is considered valid only for the current session and is erased upon closing the web browser.

```
<META HTTP-EQUIV="Set-Cookie" CONTENT="cookiename=IAM;expires=Monday, 19-
Sept-99 12:44:59 GMT; path=/">
```

- Window-target
  This one specifies the "named window" of the current page, and can be used to prevent a page from appearing inside another framed page. Usually this means that the web browser will force the page to go to the top frame set.

```
<META HTTP-EQUIV="Window-target" CONTENT="_top">
```

- Keywords and description
  These allow the search engines to easily index your site pages using the keywords you specifically tell it, along with a description of your site. The "keywords" attribute tells the search engines which keywords to use, for example:

```
<META NAME ="keywords" CONTENT="LotusScript, JavaScript, Java, Corba">
```

"Description" lets you add your own description of the page for the search engines to use:

```
<META NAME="description" CONTENT="This page is about the various programming
languages that the Domino supports.">
```

Make sure that you use several of your keywords in your description.
Tip: When using the keyword and description, you can have the same description enclosed in the comment tags so the web search engine crawlers, spiders, and robots can obtain your keywords to update their search engine index for faster search. To do that, just use the regular comment tags, like this:

```
 <!--// This page is about the various programming languages that the Domino
supports. //--!>
```

Understanding the evaluation order of programmable objects in form on the web

Before you add programming to a web application, it is helpful to understand the order in which Domino checks the programmable objects when Domino opens and saves editable documents.

Open sequence
Domino follows this sequence when web users open an editable document or create a new document:
1. Evaluate formulas in one of these ways:

    - If the database property "Use JavaScript when generating pages" is selected, Domino evaluates all formulas that are not hotspot or button formulas, including default value formulas, computed subforms, computed-for-display ,and computed fields.
    - If "Use JavaScript" is not selected, Domino evaluates all formulas, including hotspot and button formulas, default value formulas, computed subforms, computed-for-display, and computed fields.

2. Run the Open agent (called from a WebQueryOpen form event or a $$QueryOpenAgent field) if it exists.
3. Evaluate Hide-when formulas and conditions.
4. Convert the document to HTML.

Save sequence
Domino follows this sequence when web users save an editable document:
1. Evaluate all default value formulas.
2. Replace editable field values with any values sent by the user.
3. Evaluate all input translation formulas and re-evaluate computed and computed-for-display fields.
4. Evaluate all input validation formulas.
5. Run the save agent (called from a WebQuerySave form event or a $$QuerySaveAgent field) if it exists.
6. Save the document in the database and mail it if required.

7. Evaluate the $$Return field -- if it exists and if a save agent did not produce output.
8. Display output to the browser:

- If there is a save agent that returns output, return that output to the browser. The output could be an HTML page created by the agent or a page to which the agent redirects the user by issuing a URL.
- If there is a $$Return field that returns output, return that output to the browser.
- Otherwise, return the default message "Form processed" and return to the view the user came from.

Improving performance when redirecting users to another Domino URL

When a $$Return field or a WebQuerySave agent generates a URL to display the next appropriate page or document, such URL is known as a redirection URL. The normal sequence of events for processing redirection URLs is:

1. The browser submits the HTML form to the Domino server.
   2. The server evaluates the $$Return formula or runs the WebQuerySave agent to produce the redirection URL, for example:

```
"[mydb.nsf/view?OpenView]"
```

3. The server sends the redirection URL back to the browser.
4. The browser accepts that URL as input and makes the request for the URL from the server.
5. The URL goes back to the server to process the request.
6. The server sends the result of that request back to the browser.

The process described above involves two round trips from the browser to the server. If the redirected URL references a database on the same Domino server, you can avoid the extra trip back to the server (and therefore, improve response time) by using double brackets "[[ ]]" around the URL. The server recognizes this as an internal redirection URL and the transaction sequence becomes:

1. The browser submits the HTML form to the Domino server.
2. The server evaluates the $$Return formula or runs the WebQuerySave agent to produce an internal redirection URL, for example:

```
"[[mydb.nsf/view?OpenView]]"
```

3. The server recognizes the URL as an internal redirection URL, evaluates the request, and sends the results of that request back to the browser.

Using CGI variables to collect information about users

Common Gateway Interface (CGI) is a standard for connecting external applications with HTTP servers. When a web user saves or opens a document, Domino stores information

about the user in CGI variables. If a field exists with the name of a variable, Domino passes the information to the field. For example, a field named Remote_Addr uses the Internet Protocol (IP) address of the user as the field value. The ability to use CGI variables is useful in computed-for-display fields and for determining hide-when conditions based on CGI variables. Hide the field during editing, so users cannot enter information in it.

Tips for making maintenance easier

Maintaining a large application can be time-consuming. Here are some tips that can lighten the burden:

- In Domino applications, the webmaster, administrator, or system support who must keep the lists or keywords up to date should have the Editor Access authority or appropriate roles in the database ACL for editing the documents in your look-up database. They can then add, delete, and modify the lists that your @Db functions utilize. This helps to save time of development team to maintain the lists in the application or to update the hard-coded lists.
- Minimize the number of design elements. Do not create redundant or duplicate design elements because as the number of elements increase, the maintenance cost increase too. For example, a change in business rules that requires to modify a complex field formula in several forms and views. Instead, always have a practice to use shared fields, shared columns, functions in script library, subforms, and other shared resources to minimize the maintenance efforts.
- Document formulas: The complex formulas in the form events, fields, actions, hide-when and so on, should be written with proper spacing, with tabs and new line to make the formula eminently more readable, easy to understand, and easy to maintain.
- Document fields: Use hidden text in the form to add a couple of lines just above or below each section of your form to document what the fields in that section are used for.
- Restrict access to groups: Create group names and place them in the public directory and in your access control list (ACL). When granting access to someone new in your company, simply add their name to the group record in the Domino directory. There is a field called Description which can be used as a reminder as to why someone has been granted access.
- Limit delete access to reduce the number of deletion stubs. In most applications, documents should not be deleted frequently. If users have to delete documents from your application, find out why. Make sure that the users understand well how to use your application by providing good training and documentation. Also make sure that the document deletion is not a result of a design flaw.
- Take the advantage of the Help to make the application self-documenting. Create field help. Use the Help About and the Help Using documents. If necessary, create a Help view and Help documents.

4.5.2 View design and coding considerations

When designing views, keep in mind that there are several steps that you can take to increase performance.

First, try to use less complex formulas. For example, performing calculations on field values in column formulas is not as efficient as doing the calculation in a field on a form,

thus, making the column formula very simple. The same holds true for selection formulas, and especially for selection formulas that use Time and Date functions, which render the view index out of date constantly. Reducing the complexity of the formulas contained in a given view design can significantly improve the performance of that view. The more documents in a view, the more noticeable change you are likely to see.

Unread marks have a performance impact on views. Try to use unread marks only in those views that are really needed. If you do have to use them, remember that using the option "Unread Documents only" is more efficient than "Compute and Display at all levels of View".

Restricting readership of individual documents by using a Readers field or Privileges also has an impact on view performance. (Restricting access to the entire view has no impact). In case of Notes client based application, if you must restrict access to documents, use views such as "Shared to Private on first use " that can select only those documents that can be seen by the user, instead of collecting all of the documents for all users in a single view.

Every time a document is composed, deleted, or updated, Notes has to flag every view whose selection formula contains that document. Those views are now out of date, and have to be refreshed by the next user who enters the view or by the Domino server when it next indexes the view. The more views that require update, the more often a user will be in the position of refreshing a view when opening it. Also, more views means more time for the indexer to run when it updates the indexes of all views in the database. **Note:** Views and folders that must be refreshed to remain current. The more views you have, the more work is required by the server to refresh these views.

It is possible to split a database onto multiple servers, and have different views on the different servers. For instance, the read-only views could be on the database of the Statistics, Printing, Searching, and Manager replica. The working views could be on the database of Users replica. The two replicas would see each other as Editors only, so that they would not trade view design. The Users database would constantly be updating because of user traffic. However, with the Managers' views taken out, there would be far less updating to be done. Furthermore, managers are not even accessing this copy of the database, so that eliminates the authentication process of that user group. The Managers' copy would only be updated after replication. Because the Manager server probably has a much smaller user base and thus less authentication and indexing are required, you can set the replication of the User and Manager databases to take place only on the Manager server and run indexer only after the replication cycle.

You also can set up the views as 'Shared', 'Private on First Use', or 'Private'. For these view type, the indexes are stored locally on the users' desktop.dsk files. hese views have slower performance than regular shared views. Depending upon the size of the view index, the users might find that their desktop.dsk files grow alarmingly.

Minimizing the complexity of views

IBM Lotus offers great tools for tailoring views. Use these facts to help you make decisions about what kinds of views to make:

- Minimize the number of columns with formulas. It is more efficient to have a formula calculated in a field in the documents, and then simply display that field value in a view column.

- Sorted columns and categorized column take longer to index than non-sorted columns. You might experience a noticeable slowdown in indexing time after adding a sorted or categorized column to a view that had no sorted or categorized column. Each additional sorted or categorized column increases the indexing time, though progressively less.
- The size of the view index has a direct relationship to the number of columns and the number of documents.
- Documents appearing in multiple categories increase the size of the view index considerably. For example, for the users who use All By Category and All By Person views, the view index mail file size of All By Person view categorized on all recipients is significantly larger than the mail size of the All By Category view index.
- Avoid using form formulas whenever is possible. Computed subforms are the faster alternative. However, consider the maintenance aspect in case of the complex form structure and the number of fields in the form.
- The more information contained in the SUMMARY fields, the larger the view index.

Refer to 6.1 View index performance considerations for more specific information on view index performance.

Structuring view selection formulas to improve performance

The way the selection formula of a view is written can impact the performance of your application. If all documents in a view use the same form, for example, use that form name in the view selection formula. Notes scans only the documents created with that form, not the documents created (or edited and saved) with other forms.
For example, in a view with hundreds of documents, but only a few that include the field "Company", the formula

```
SELECT form = "Main" & company = "Acme"
```

would be more efficient than

```
Select company = "Acme"
```

if the "Company" field exists only on the "Main" form. This will provide the biggest performance gain when the selected documents represent a small subset of the entire database.

Tips for developing web views

Although you can include a view or folder in a web application, you have little control over its presentation in a browser. Views or folders always display in a full screen with default navigation buttons.
You can regain some control over how the view displays if you embed it in a form, page, or document. You can maintain the same functionality available in Notes applications, and you can control the size and appearance of a view or folder display. You can also

combine view and folder panes with other form elements (such as styled text) and graphics.
When you embed a folder pane, you show a list of views and folders with the standard Folders navigator, as seen in Notes with the **view - show folders** command. The font style you apply to the field applies to the style used to display the folders.
If you have multiple views or folder panes you want to display in an application, you can create one or more view templates that control how the embedded objects display. A view template is actually a specially named form that provides the design structure for displaying the embedded folder panes or views in an application.

- Use a form as a view template to customize the view and make it look more web-like.
- Use only minimum required columns for the web views.
- By default, views show 30 rows at a time. You may want to vary this number by using the "Default lines per view page" setting in the Domino server document of the Public Address Book.

Use these tips to improve view performance on the web:

- Avoid cascading views because the screen has to be redrawn and the page appears to jump.
- Remove all unnecessary columns.
- Do not use form formulas to display documents to web users. Instead, use computed subforms that are hidden from Notes users to show Web-only information.
- Avoid time-sensitive column formulas that use @functions such as @Now, @Created, and @Modified. Domino generates web views as HTML pages on the fly. Time-sensitive formulas recalculate each time a web user takes an action in the view.
  For example, if you want a "What's New" view for a web site, do not create a view selection formula that checks to see if every document's created date is within seven days of @Now. Such a formula would force the view to be dynamically regenerated every time the user interacts with it. Instead, in each document, set a flag in a hidden field to either a 1 or a 0 depending on whether the created date falls within seven days of the current date. (This is the most convenient approach, but could result in the unnecessary replication of documents as the flag gets reset. If the database is being replicated, you might use a Lotus Script agent to move documents in and out of a folder that holds only the most recent documents.)

Scrubbing a view affects performance

Active databases that let users create and reassign categories to documents can accumulate empty categories that contain no documents, yet clutter categorized views. You should be aware of the performance implications of reducing this visual clutter. Selecting "Don't show categories having zero documents" on the View Properties box causes Domino to take additional steps to test each category in the view to determine whether it includes documents. As an alternative to adding processing each time a view is refreshed, consider proactive removal of empty categories with a timed agent, or tighter control of the creation of categories in the database.

Use single-activity view

A single-activity view helps users focus on the tasks for an occasional activity, such as archiving or printing, and maintains instructions in one place. Use the view action bar to

walk the user through all necessary tasks the same way you walk a user through the process of filling out a form. To enhance the view, create a companion navigator that includes a text object containing instructions.

Predetermine view categories

Applications that allow users to specify view categories become confusing when users add too many categories and do not keep them consistent. It is a good practice to plan categories in advance.

When creating the Categories field, enter a text list of categories or use a look-up formula to provide categories. To give users some flexibility, enable the "Allow values not in list" option so that users can add custom categories.

Improving view and folder performance

- Reduce the number of views and folders to improve database performance.
- Avoid writing selection formulas or view column formulas based on a date calculation. This causes performance lag because the calculation must be done every time the view is opened. For example:

```
SELECT @today<= @adjust(@created; 0;0;3;0;0;0)
```

- Minimize the number of columns that contain formulas. Instead, place the formula on a form, and store its result in a field to be displayed in the column.
- You can select "Hide detail rows" to suppress subtotals for each category and sub category to display totals without the clutter of extra numbers. Totals calculate only for main documents; response documents are not included. Because column totals recalculate each time you open the view, they may affect database and overall server performance.
- Do not allow documents to appear in multiple categories. When a document appears in multiple categories, the size of the view index increases, causing the view to open and refresh more slowly.
- View indexes can affect performance. Examine your view indexing options to see if you can improve performance. For example, you might refresh the view less frequently.
- Compact databases and purge deletion stubs regularly.
- Increase the setting for the NSF_BUFFER_POOL_SIZE variable in the Notes.ini file. This setting specifies the maximum size (in bytes) of the NSF buffer pool, which is a section of memory dedicated to buffering I/O transfers between the NIF indexing functions and disk storage. The setting, which is determined automatically by the server, is by default is 1/4 of your physical memory. You can increase this setting up to 1/2 of your physical memory to increase the cache for handling large views or many views.

Using $$ViewTemplateDefault as a deterrent

Almost every Domino application uses hidden views for look-ups and other necessary events and processes. These hidden views cannot be secured to prevent user accessing had the user knows how to reconstruct the Domino URL to view the data. For this reason, it is a good practice to use the $$ViewTemplateDefault form as a shield to deter users who might attempt to gain access to the areas of your application that you do not want the user to see. This technique requires a little extra work from the developers because for every view in your application, there is a view template required. On your $$ViewTemplateDefault variable, it is possible to capture the names of the users who are attempting to access your application, by using hidden fields that calculate the @Username formula or some CGI field value.
Caution: This technique is *not* a true Domino security implementation.

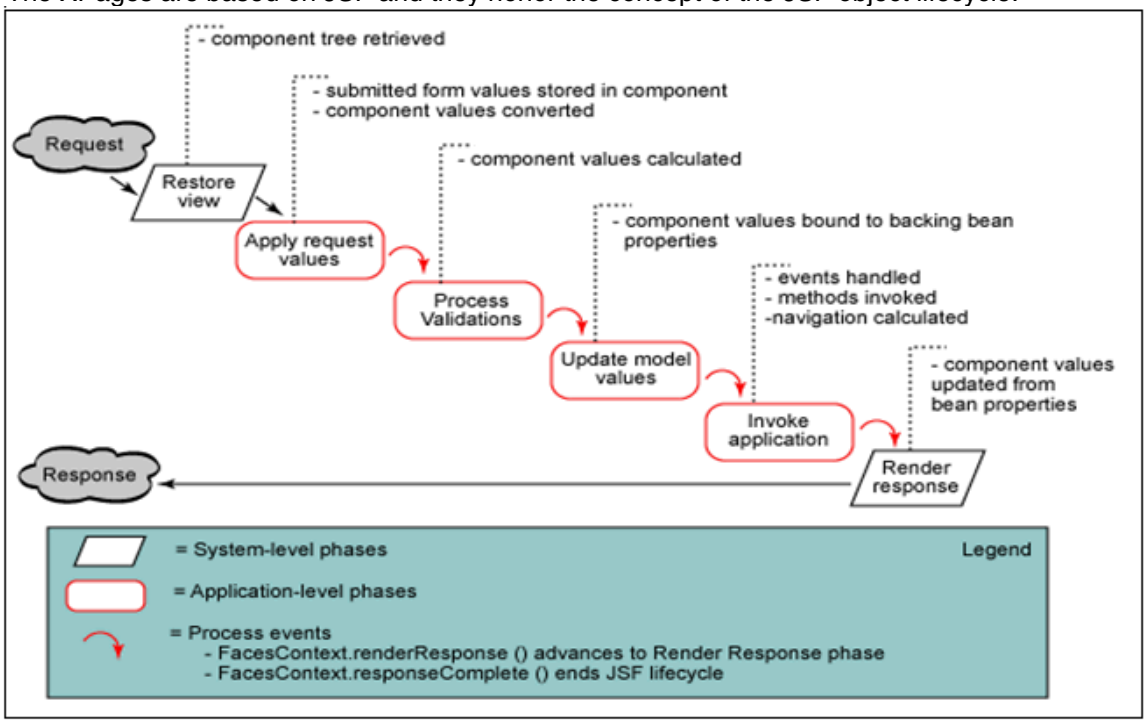
Make DHTML views for web users

It is the best to use the XPages view to provide rich Web 2.0 features that are faster and easy to build with better GUI. However, if you are not using the XPages, then consider DHTML views as an option for displaying views in a web application. This type of view consists of an embedded view surrounded by DHTML code. Follow these steps to create a DHTML view:

1. On the Advanced tab of the view properties box, in " Web Access" section, select **Treat view contents as HTML**.
2. The view should have the column formulas with embedded HTML code that use field names to provide custom user interface to the view.
3. The view is then embedded in a page or a form that contains the DHTML code. Mark the text as pass through HTML by highlighting it and selecting **Text - Pass-Thru HTML**.

4.5.3 XPage design and coding consideration

The XPages are based on JSF and they honor the concept of the JSF object lifecycle.



Here are some of the rules that you should follow to obtain the maximum out of your application.

Upgrade
Upgrade your environment to the latest available version.

Custom controls
You should use custom controls wherever possible in your application for effective reusability of the code. Refer to 4.6 Optimizing XPages using custom control for more details.

Themes
In order to have a hassle free and simplistic user interface control, you should use themes which gives you a centralized control of all the user interface classes. Refer to 5.1 Themes for more details.

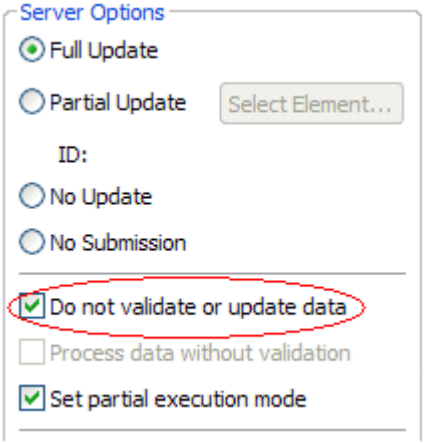
GET-based requests
Use GET-based requests wherever possible instead of POST-based requests. This saves valuable server processing time and also allows you to skip phase two to five of the lifecycle.

Partial execution mode
Use "Partial Execution" mode if only a portion of the page should be updated and processed. You can use "execMode" property of the event handler. If you combine this with "execId" then you can specify a particular target on the page. This allows phases 2, 3, 4, and 5 to be processed on the submitting control, and any children referenced by the event handler.

Readonly property
When there is no processing required by any controls within a container, use the "readonly" property on container type controls (For example, on a panel containing a list of computed field controls, nothing is editable; therefore, no server-side event handlers need to be executed).

Partial refresh
Use "Partial Refresh" whenever possible. It improves the performance of an application by reducing the amount of HTML markup that must be processed which in turn leads to saving in the server utilization time. This will improve performance of phase 6 of the lifecycle.

Immediate property
You can eliminate lifecycle phases 3, 4, and 5 using the "immediate" property. Occasionally, you only need to identify the triggered event handler and then redirect to another XPage without any further server-side processing against the underlying data model. In this case, you do not need the Process Validations, Update Model Values, and Invoke Application phases to be executed. One example of this type of interaction is a Cancel button that should navigate back to the previous XPage, or some other XPage. Use this option when your event handler needs to perform an action and then redirect to a different XPage.



Data Caching
When a xp:dominoView data source is included on an XPage, the server has to save the values displayed by the page in case of a POST-back request. This ensures that the same data is available to all the requests in the lifecycle regardless of the changes made in the underlying view. However, this has its side-effects:

- Data saved for the page consumes a large amount of memory which overloads the JVM on the server.
- Extra processing as some objects cannot be kept in memory between requests.

To avoid this you can use ‘dataCache’ property to optimize the view. There are three values supported:

- Full [default]: The entire view data is persisted after a request. Consumes most memory and CPU utilization.
- ID: Only a minimum amount of scalar type view related data, such as ID and position, is persisted after a request. Consumes least amount of CPU utilization and optimized amount of memory utilization.
- None: No view related data is persisted after a request. Consumes least amount of memory utilization and most amount of CPU utilization.

Value bindings
In your XPage's value bindings, you can utilize one of two different binding types to control when the expression is evaluated. These binding types are:

- #{ }: Compute dynamically
- ${ }: Compute on page load

The first, #{ } (hash-brace), tells the XPage to evaluate the binding on page load and every subsequent refresh. The second type, ${ } (dollar-brace), tells the runtime environment to only evaluate the value at page load and use the same value for the life of the page. You should be using the dollar-brace if your value will not change throughout the life of your XPage.

Scope variables
You should use the viewScope object to maintain server-side buffering objects and variables for an XPage instead of the other scopes, such as sessionScope and applicationScope which use more memory. However, you should also look at the usage of the referred object to decide its scope.

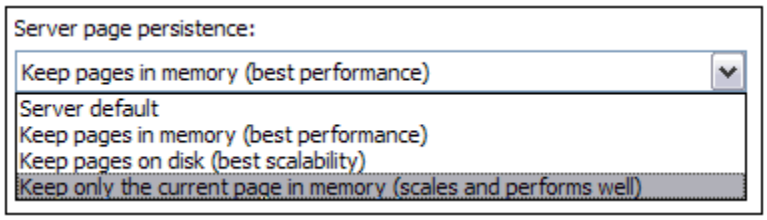
- viewScope: Exists for the life a view (rendering of a page)
- requestScope : Exists for the life of a request (can span pages)
- sessionScope : Exists for the life of user session
- applicationScope: Exists for the first loading of the application until a timeout (default is 30 minutes)

Use Java
Minimize the use of SSJS (server-side JavaScript). Every time any SSJS expression is evaluated, Domino parses the expression in realtime into an abstract syntax tree, then runs Java code that is a rough approximation of how the JavaScript specification states that code matching the detected syntax should run. The more you move your logic directly into true Java classes, the faster it runs.

Memory utilization
The *notes.ini* parameter of "HTTPJVMMaxHeapSize" on the server gives you control over the amount of memory allocated to the JVM. You should work with your administrator to have it set to 512M or 1024M to serve more concurrent users depending on the availability of resources on your server. In case of 32-bit systems especially

Windows you should know that there is only a limited amount of memory available to you.

Server page persistence
The JSF framework of XPages utilizes persistence for a predefined maximum number of component tree state representations for XPages requested by a user during their session. This process happens in order to allow a component tree to be restored with its previous state as a user navigates through an application. It also helps to improve the performance of the phase 1 of the JSF lifecycle when reconstructing a component tree. This feature is known as Server Page Persistence and you can configure it using the XPages tab of the Application Properties.


Server page persistence:
Keep pages in memory (best performance)
Server default
Keep pages in memory (best performance)
Keep pages on disk (best scalability)
Keep only the current page in memory (scales and performs well)

Preloading
In 8.5.3 new notes.ini parameters have been introduces which allows you to preload underlying XPages Java classes before the application is opened. They work both on client and server so the user has the same experience on both platforms.

- XPagesPreload=1
- XPagesPreloadDB=db.nsf/myPage.xsp,other.nsf/myPage.xsp (for specific applicaitons)

4.6. Optimizing XPages using custom control

If you are a Domino developer, you are already familiar with the practice of using common design element subform in a form. If you have been a Java developer, you must have used extensions of other classes in your code.

Custom Controls are a similar new design element that was introduced along with XPages in the Domino Designer. In short, Custom Controls act as reusable subcomponents of XPages.

Some of the characteristics that define Custom Controls are:

- They are componentized parts of your application.
- They are much advanced version of subforms.
- You can use them multiple times on a single page.
- You can define parameters for custom controls.
- They can be shared across applications.
- They can be nested.

There can be different approaches for using Custom Controls in your application. The first and the most typical approach is to use them for coding parts of application that contain some logic and are used multiple times across the application. The second

approach is to use them for breaking down the application user interface in parts so they can be used and changed easily across multiple pages. For example, when you are using the OneUI theme, you break down the components under different custom controls for heard, footer, left column, content, and so on .

The use of Custom Controls with XPages, can simplifies your development effort to a great extent, the sooner you start using them the better it will be. Custom Controls allow you to break up your application in small manageable parts which can be reused as part of the bigger picture. This approach is particularly beneficial if there are multiple team members working on the same application. You are also able to take advantage of other sources internal or external if you use this approach and this in turn help you gain in productivity and efficiency.

4.7. Naming standards

The purpose of naming standards is mainly to enforce consistency and its immunity to remain compatible with the environment. Naming standards should be decided for each and every resource in the application including the database name, form, view, agent, and other design elements.

In additional to the name, make use of alias names wherever possible. This allows you to change or translate the name without breaking the reference to the original name in the application design.
If your development team already uses a more elaborate naming standard where aliases for different design elements have different prefixes, you can stay with that naming standard. The important thing is to have a consistent standard and to apply it from the very start of the development cycle.

The following are the general guidelines about naming:

- Do’s
    - Consistent naming conventions across design module.
    - Name should identify an item or its purpose.
    - Use of prefix to group the like items.
    - Use of cascading names to provide a hierarchy representation to group elements (that is, use “\” backslash).
    - User interface elements should be descriptive and accelerator key character should be identified.

- Don’t’s
    - Long names should be avoided.
    - Special characters usage to be avoided, for example, “$”, “#”, and “@”.
    - Special keywords should be avoided.

4.8. Internationalization

Every application should be designed with international users in mind. Though your target audience might be entirely in one country initially, the situation might change in the future. Some applications grow over time, and some application designs are recycled into new applications that might have different target audiences. This section describes some code techniques for dealing with international applications.

This article is not meant to be an all-encompassing detail of how to internationalize an application. There are many resources available on the Internet to assist with further details of the items discussed below.

4.8.1 Dates and times

When displaying dates, be aware that not everyone views a date value the same. For example, 10/11/12 could mean the 10th of November or the 11th of October. It is best to show dates with a month name (either abbreviated of spelled out). It is also best to always use a four-digit year, to decrease confusion.

To format a date in the current user's locale, there are built-in Java classes to do this. As an example, this code shows the current date (“now”) and current time in the user's defined LONG date format:

```
java.text.DateFormat.getDateInstance(java.text.DateFormat.LONG,
context.getLocale()).format(now)
java.text.DateFormat.getTimeInstance(java.text.DateFormat.LONG,
context.getLocale()).format(now)
```

If designing a Notes client application, take advantage of the client's capability to format dates according to the user's local preferences. This mainly means that you should avoid converting date/times to string values that you then store in the database. Dates and times should be stored using the built-in date-time item type, and only converted by the user's client.

As an example of what not to do, consider the following field formula for a computed field:

```
"Updated by " + @V3UserName + " on " + @Text(@Today)
```

If a document is edited by a user in France, let's say, then the field is assigned a value such as "Updated by Michelle Dubois on 7/4/2012", meaning April 7th. But users in some other places -- the USA, for instance -- are used to seeing the month first, so they are likely to misinterpret this as July 4th. What's worse, because the value in each document reflects the settings of the user who last edited it, the documents are not even consistent in how they format dates; in some, "7/4" means April 7th, in others it means July 4th.

To avoid such difficulties, your application should store this information in two fields -- a Text field for the name of the editing user, and a date/time field for the date. Since the client calculates how to display the date when the user opens the document, the user's preferred settings will always be used. Even if you override the default formatting to specify that dates will always be displayed in U.S. format, at least then your formatting will be consistent.

A similar situation occurs in view column formulas, when @Text is used to convert date/times. Because the view index is calculated by the server, the conversion takes place according to the server's settings rather than those of the end user. It is generally better, if your column is to display a date or time value, to have a view column formula that returns a date/time value, not text. You can use the column settings to control formatting if you wish, but it's usually better to let the users see dates in the format that they have established as their personal preference.

NOTE: While you would want to avoid using @Text to convert a date in a computed field or computed when composed, it's fine to do this in a computed *for display* field, since the value in that case is not stored, but only used locally.

NOTE: in the above example, it's also a bad idea to make the text "Updated by" and "on" part of the field value, since that will cause problems if you make the application multilingual later, and because it's a waste of space to store these characters in every document.

4.8.2 Time zones

When displaying time values, you should make it clear what time zone they are in. Either display the zone, or if you convert all times to the local time zone, indicate this somehow (e.g. by the text "(local time)" after the value. A time of 8:00 AM means quite different things if it is 8:00 AM Greenwich Mean Time or 8:00 AM Hawaiian Standard Time. The Edit Box controls in XPages and the Date/Time fields in forms and views can easily show the time zone along with the time value. There is a TimeZone field type which can be used to select a time zone in traditional Notes development. A TimeZone field converts to a combo box with all the available time zones on the web.

In XPages development, there is an application property which says if the time zone should use the server default (choice comes from a properties file), the user's time zone, or the server's time zone. There is also a TimeZone object in Java that can be used. For more information about using time zones in your XPages development, refer to the wiki

article Time Zones in XPages

Another zone issue can result from using the LotusScript date/time variant datatype to assign date-only values to document fields. For details, and general recommendations and troubleshooting for working with date/time values, consult the Designer wiki document, Working with Date/Time Values in Lotus Notes.

If you need a particular component of a date -- if you want just the year, for instance -- do not convert the date to text, then use string functions (Mid$, @Explode, etcetera) to find the year, month or day. Your code will break when used by someone with different preferences than yours -- which can occur even if the user is in the same city.

4.8.3 Currency

When developing international applications, you cannot make assumptions about number and currency formats. For example, the thousands separator in Europe is a period and the separator between whole amounts and fractional amounts is a comma. But in the United States, the comma is the thousands separator and a period separates the whole

amount and the fractional amount. There are many JavaScript examples on the web that will format a number in the United States format. The following best practice can be used to display a currency value in the end user's format:

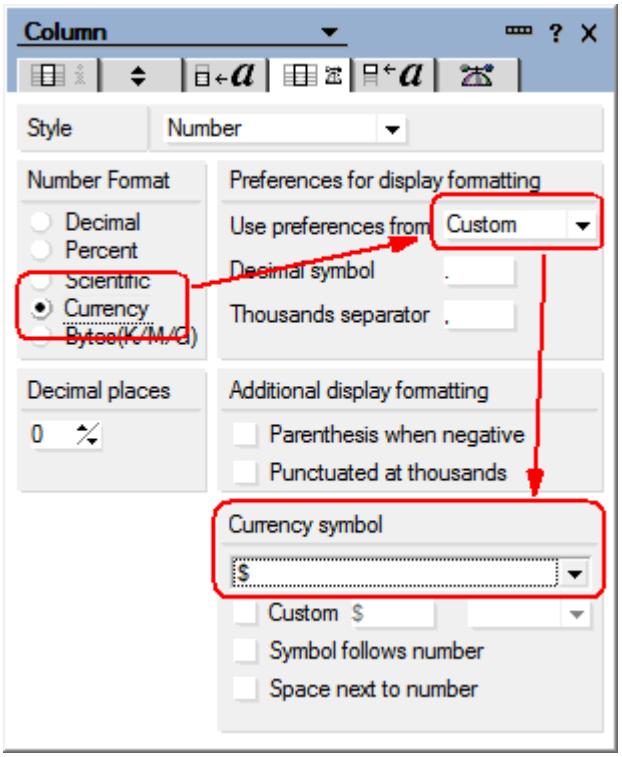
- Determine if the user has a decimal or comma as the separator between the whole amount and the fractional amount
- Use the “other” character as the thousands separator
- Format the number appropriately

To determine the separator character between the whole amount and the fractional amount, use the following server-side JavaScript code:

```
var decimalSep = Number("1.5").toLocaleString().substr(1,1);
```

Concerns of the decimal and thousands delimiter being different with different users also apply when developing Notes client applications. As explained above regarding date/time values, when working with numbers, it's better to store the number in an item of type Number, not as text. Avoid using @Text in computed field formulas and view column formulas.

If displaying a value with a currency symbol, bear in mind that the local currency symbol is an OS preference also. If your value is in US dollars (let's say), make sure you specify $ as your currency symbol in the formatting settings of Notes fields and view columns. Otherwise, users in other countries will see their local currency symbol beside the number.



There is not an value type in Lotus Notes that includes an amount *and* a currency selection; you would have to store this information in two separate fields.

4.8.4 Native language

When possible, application output should be available in a variety of languages. This broadens the audience for your application and decreases confusion for uses who do not speak your language fluently. The wiki "XPages: How to use the localization options" describes how to use multiple languages for your XPage applications.
For Notes client applications, Domino Global Workbench (DGW) provides a way to convert a single-language application into multiple languages. It extracts strings and creates properties files which you can hand over to translators; then you use DGW again to import the files to create different national-language versions of each design element in your application. So for instance, your one NSF would contain a form in English, and the same form in French, and so on. Notes client users will be presented the version of the form, view or whatever that corresponds to their language preference.

NOTE: Views have performance impact, as described elsewhere in this guide. Do not use the "Automatic" indexing option for views in multilingual applications, since that will cause all language versions of the view to be indexed on all servers, even if nobody on the server uses that language. "Auto on first use" is a better choice. You might also want to discard indexes after a shorter number of days than you otherwise might.

4.8.5 The I18n class in XPages

You can use the I18n class to perform many types of conversions and result in correct displays of information based on the locale of the reader. The following example displays a date and time in your desired format:

```
I18n.toString(JavaDate, pattern)
```

There are many date patterns available.
The I18n class can also be used to convert a time from one time zone to another time zone. For example, you can display the time in both Greenwich Mean Time and the user's current time zone.

4.9. Scoped variables

If you are a Domino developer, you are already familiar with the practice of using common design element subform in a form. If you have been a Java developer, you must have used extensions of other classes in your code.

Custom Controls are a similar new design element that was introduced along with XPages in the Domino Designer. In short, Custom Controls act as reusable subcomponents of XPages.

Some of the characteristics that define Custom Controls are:

- They are componentized parts of your application.
- They are much advanced version of subforms.
- You can use them multiple times on a single page.
- You can define parameters for custom controls.
- They can be shared across applications.
- They can be nested.

There can be different approaches for using Custom Controls in your application. The first and the most typical approach is to use them for coding parts of application that contain some logic and are used multiple times across the application. The second approach is to use them for breaking down the application user interface in parts so they can be used and changed easily across multiple pages. For example, when you are using the OneUI theme, you break down the components under different custom controls for heard, footer, left column, content, and so on .

The use of Custom Controls with XPages, can simplifies your development effort to a great extent, the sooner you start using them the better it will be. Custom Controls allow you to break up your application in small manageable parts which can be reused as part of the bigger picture. This approach is particularly beneficial if there are multiple team members working on the same application. You are also able to take advantage of other sources internal or external if you use this approach and this in turn help you gain in productivity and efficiency.

Chapter 5. User interface considerations

When designing an application you must remember that people will be interacting with it through the front end you design. If you make it hard for the users to understand what they are to do or make it confusing, then they will try at all costs to not use it. Internal application suffer the most since it is to a captive audience. Make the user interface clean and intuitive for the users (not necessarily for the developers). Pick only a couple of font families, the fewer the better. Target the age of your audience when picking font size.

XPage applications are web application that can also be utilized in the Notes Client. There is much information regarding the proper design of a web application and this chapter deals with the aspects that pertain to XPages.

Make sure your design:

- Is intuitive and easy to understand.
- Is Easy to use.
- Provides benefits to the user.
- Is readable: no small type or hard to read colors.

The benefits of a good user interface include

- Increase employee productivity.
- Decrease employee error.
- Increase employee buy-in and participation in the application.
- Reduce training costs.
- Decrease technical support.
- Increase money saved.

When designing user interface, the bottom line is that it is all about the user. Make the experience positive for them and successful for your company.

5.1. Themes

Themes were added with XPage to provide the developers with the ability to define the look and feel of an application. The themes provide the following abilities:

- Extensible.
- Add CSS resources to an XPage.
- Add JavaScript resources to an XPage.
- Link to specific controls using its theme ID.
- Apply property values to controls.
- Easy to select a different theme to give a different look and feel.
- Can be set globally (all applications) or per application.
- Different themes can be applied depending on the context.

The theme is handled on the server side where it drives the page creation and controls the HTML generation. It is fully used by XPages, partially by the "classic" web engine.

The hardest part will be recognizing how many things you can control from a theme if the results are to be used across multiple applications then put it in the theme.

For the best practice, using themes make your application easier to change, and, make it easier to develop and maintain consistent elements that themes are applied to.

5.1.1 Themes are extensible

Themes can inherit from other themes and override elements of the inherited themes. You normally create a theme by extending one of the standard themes. Domino comes with three themes:

- Webstandard: The basic clean theme
- OneUIv2 (oneuiv2, oneuiv2_red, oneuiv2_green, oneuiv2_metal, oneuiv2_gold)
- Notes: Notes client theme

The default is the Webstandard.

5.1.2 CSS resources

Creating multiple theme document allows you to build different sets of style sheets that can be selected when the application is deployed. The application can easily have the look and feel changed by selecting which theme to use. For example, you can have different color-based themes. You can choose between red, blue, gold, or metal color themes with OneUI CSS framework.

There are two options for applying CSS style sheets to an XPage applications:

- Add the CSS style sheet as a resource and select the styles in the XPage or custom control.
- Create a theme document and reference the CSS style sheet from it.

Adding the style sheet directly to the XPage or custom control does have a few advantages:

- The list of classes will show in the Styles section for all the controls on the page.
- The design perspective for the XPage will try and apply the styles so visually you will see how the page will look without having to preview it.

Note: You can add the style sheet to a custom control so it is applied to the rest of an XPage. However, make sure that you do not have it as a resource on multiple controls because it is loaded each time it is listed.

Some of the advantages to using themes include:

- Building applications that are easily customized by a third-party.
- Tailoring the site to the user: You can compute the styling by requested device, language, or browser.
- Inherit from other themes such as the built in WebStandard or OneUI.
- Settings that are higher up the tree (themes you have extended from, styling set within the control) that can be overridden.

- Stylesheet and styling information can be computed rendered="#{javascript:context.isDirectionRTL()}"
- Component styling can be globally controlled; no need to set style information or class. All buttons across all applications have the same style with one place to control all the edits.

The disadvantage is that the list of classes in the style sheets is not display in the Styles section for each control and the design perspective will not render the controls with the style sheets. This can fixed by using a dummy css files that only contains the style class names. For example:
.lotusBtn {}
.lotusBtn:focus {}
.lotusBtn:hover {}
.lotusBtnAction {}

This allows the styles to show in the list and to be rendered in the Design pane. Do not forget to remove it before deploying so you save resources.

5.1.3 Building theme

To build a theme, the first thing is to select the theme you will be building upon by setting the inheritance in the Theme root element:

```
<theme extends="oneuiv2">
```

If a theme extends a theme with the same name, then the new theme is searched at the upper level. For example, a database theme "oursite" can inherit from a global theme "oursite". All *circular references* are checked and prevented.

Themes can control more than styling, there are other properties that can be set or overridden. Themes can be extended by using JavaScript, such as getting the type of browser and applying the style sheet needed. This allows for a style sheet to support older browsers or mobile devices and have the selection based on the type of device. This can be done by adding a "rendered" attribute to the resource tag. The following is an example:

```
<resource rendered="#{javascript:context.getUserAgent().isIE(0,6) == true}">
```

The rendered attribute is running server side JavaScript to determine which browser is being used in this example. If it is Internet Explorer (IE) versions 0 thru 6, the resource is rendered to the web browser. The resource does not get sent to the user's web browser if you are using higher version of Internet Explore or a different web browser. This is a great way of adding special style sheets that contain hacks for some of the behaviors of styles in earlier versions of IE.

Check for Safari If you want to include HTML5 and support the webkit based browsers: rendered="#{javascript:context.userAgent().isSafari()==true}"

Style classes can be set depending on the scoped variables, as in the example below:

```
<control>
        <name>Link.collapse</name>
        <property>
                <name>styleClass</name>
                        <value>#{javascript:(sessionScope.ec == null ||
sesionScope.ec == 0) ? “xspDisableCollapse”: “xspEnabledColapse”}
                </value>
        </property>
</control>
```

The example above applies a style to expand or collapse an area based on the value of the sessionScoped variable *ec*.

You can have elements be hidden in one theme and available in another. Here is an example where additional information is display if a certain theme is selected.
Create a new theme called oneuiv2-debug that extends the oneuiv2_red theme.

```
 <!--
        The default theme contents use theme inheritance.

        Application themes can extend an existing global theme using the
        extends attribute.  Existing themes include the following options:

        1. webstandard 2. oneui 3. notes
-->
<theme extends="oneuiv2_red">
        <control>
                <name>debugInfo</name>
                <property>
                        <name>styleClass</name>
                        <value>debugInfoVisible</value>
                </property>
        </control>
</theme>:
```

The tags contain the themeID and the contains the properties to be set. So the property styleClass is set to debugInfoVisible which is a CSS class. The CSS class:

```
.debugInfoVisible {
        background-color: #EEFEE1;
        border-color: #CCEBB5;
        border: 1px solid;
        padding: 10px;
}
```

On one of the custom controls is a div tag that contains the debugging information:

```
<xp:div themeId="debugInfo"styleClass="debugInfoHidden">
          <div class="lotusHeader">
                  <h3>Debug info</h3>
                 <div class="lotusFilters">
                         Debug info is displayed when a debug-mode theme has
                         been selected in the Application Properties
                 </div>
         </div>
         <xp:label id="label3" value="isAuthor: "></xp:label>
         <xp:text escape="true" id="debugIsAuthor"
                 value="#{javascript:sessionScope.isAuthor}">
         </xp:text>
         <xp:br></xp:br>

         Author:
         <xp:text escape="true" id="computedField3">
                 <xp:this.value><![CDATA[#{javascript:var doc:NotesDocument =
dominoDoc.getDocument();
                         var from =
doc.getItemValueString("From");@Name("[ABBREVIATE]",from);}]]>
                 </xp:this.value>
         </xp:text>
         <xp:br></xp:br>

         Session Scope:
         <xp:text escape="true" id="computedField4"
                 value="#{javascript:sessionScope.values()}">
         </xp:text>
 </xp:div>
```

The div only shows if the debugTheme is selected.

You can set the control or property to over ride the base theme setting or to just extend them.

The control override value can be true (your property only is used) or false (your property is added to any in the theme):

```
<control override="true">
        <name>debugInfo</name>
        <property>
                <name>styleClass</name>
                <value>debugInfoVisible</value>
        </property>
</control>
```

The property can have the mode value be "override" (your property only is used) or "concat" (your property is added to any in the theme):

```
<control>
        <name>debugInfo</name>
        <property mode ="override">
                <name>styleClass</name>
                <value>debugInfoVisible</value>
        </property>
</control>
```

5.1.4 Themes and classic Domino

The theme property only works on classic web elements if the *Enable enhanced HTML generation*option is selected on the database properties dialog box.

To include the same style sheet on all the forms in your database, you can use a theme. First check the *Enable enhanced HTML* option. Then create a theme design element with the following content in it:

```
<resource>
         <content-type>text/css</content-type>
         <href>mystylesheet.css</href>
 </resource>
```

That automatically adds the stylesheet to the form. You can set the style applied to the Domino controls by putting the following in the theme:

```
<control>
        <name>body</name>
        <property>
                <name>style</name>
                <value>bodyframe</value>
        </property>
</control>
```

This automatically puts a style tag with the specified value on every body tag generated.

5.2. CSS: Framework and location

5.2.1 CSS – frameworks

A *CSS Framework* is a collection of pre-built cascading style sheets (CSS) designed for a specific layout and styling of web applications. These frameworks are usually well documented, configurable, and tested for popular browsers. CSS frameworks are increasingly used as a best practice for developing web applications as the benefits of using them outweigh the drawbacks.

A CSS Framework provides these advantages:

- Reduced development time: Developers are able to focus on the core functionality of the application versus the look and feel by using pre-built CSS classes for layout and styling.
- Improved user experience and reduced training needs: Due to the consistent look and feel across various web applications.
- Accommodate various browser versions and screen resolutions: You can see what browsers are support so you do not have to try to build a CSS that covers all the quirks.

- Documentation: Very few in house frameworks have documentation.

Some of the drawbacks of using a CSS framework include:

- A learning curve to understand a framework and its configuration options.
- Potentially being locked into a specific layout and navigation structure.

Some of the popular CSS Frameworks are listed below:

- OneUI (from IBM)
- Blueprint (for information on using Blueprint with XPages, see David Leedy's Notes in 9 videos at notesin9.com)
- Elements
- Grid960
- YUI 2 (from Yahoo)
- iUi (for mobile applications)

When picking a framework, make sure it supports using classes and not IDs. XPage control IDs are dynamic and created when it is rendered, so you can not be sure you have the correct ID.
Instead of using:

```
<div id="FirstName">
```

You would use :

```
<xp:panel styleClass="FirstName">.
```

Select your framework before starting to code. It is not that easy to change the layout after the application is built.

5.2.2 Using OneUI V2

All the resources for the OneUI framework (style sheets and images) are already accessible because it is delivered as part of the Domino server, Notes client, and Domino Designer. You do not have to import them separately for each application. Classes for Dojo are also included. This makes it easier to use this framework within Domino applications.

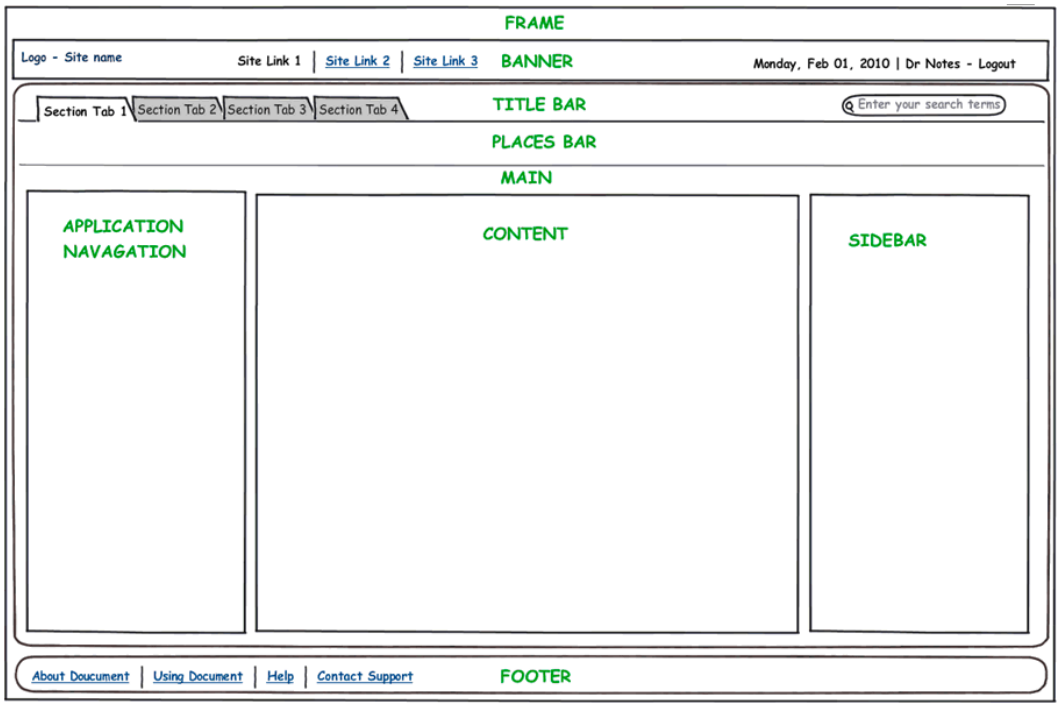
Different HTML tags that are used for web development are referred to during the lesson. All the HTML tags and attributes are supported by XPage and can easily be embedded in the code as-is. Most HTML are entered through the source view where you see the actual XML source for the XPage.

For more information, see the OneUI Developer document at http://www-12.lotus.com/ldd/doc/oneuidoc/docpublic/components/framework.htm.

Using the OneUI classes

The OneUI provides *class* to render the basic layout of the page. There are classes to

style the banner, navigation, footer content, and more.
The following figure shows the One UI breakout:



Frame
The *frame*is the main container for the rendered web page. This is where the background color and the width of the site are set.

Banner
The *banner* sits at the very top of the UI and contains application links, utility links (help, log in), and the application logo. It picks up the background of the page. It is plain white in the figure above.

Title bar
The *title bar* contains application, or sectional, navigational tabs. It also can contain a search component.

Place bar
The *place bar* sits under the title bar, to show you what "place" you are in. It usually indicates an area that you can join, such as an activity or community. The bottom border of the title bar serves as the top border of the place bar.

Main
Main is the body of the page. It contains the left column, content, and right column.

Left column – Local navigation
The *left column* is usually the navigation within a section of the application. For example, a time entry application would have a section that may take you to projects, time reports, and so on. In the Time report section, the application navigation would be to show submitted time reports, all months, and unsubmitted time reports.

Content
The *content* area is where the forms and views are displayed.

Right Column – Sidebar
The *right column – sidebar* provides additional sections that can be viewed by the user. This is similar to the left column and can be used for navigation based on how you want to layout your page.

Footer
The *footer* sits at the bottom of the page and contains links to Help and other features.

The basic HTML layout of the page is:

```
 <body class="lotusui tundra">
   <div class="lotusFrame">
       <div class="lotusBanner"></div>
       <div class="lotusTitleBar"></div>
       <div class="lotusPlaceBar"></div>
       <div class="lotusMain">
           <div class="lotusColLeft"></div>
           <div class="lotusColRight"></div>
           <div class="lotusContent"></div>
       </div>
       <div class="lotusFooter"></div>
       <div class="lotusLegal"></div>
   </div>
</body>
```

In general, when linking CSS into your page, the following order must be used:

1. Base Styles: required
2. Component Styles: required
3. Layout Styles: optional
4. Base Theme Styles: required
5. Component Theme Styles: required
6. Layout Theme Styles: optional
7. Dojo Styles (dijit.css): optional
Note: this file is from dojo's dijit and theme directory.
8. Dojo Override Styles: optional
9. Dojo Override Theme Styles: optional
10. Product-Specific Styles

For the OneUI, the CSS file order is:

1. core.css
2. defaultTheme.css
3. Dojo File: dijit.css (optional)
4. dojo.css (optional)
5. dojoTheme.css (optional)
6. your custom css

When working with any framework, break up your model page structure into custom controls.

One of the best ways to build your application is to create a custom control for each section of the framework. This way you can easily find the section to change and also can:

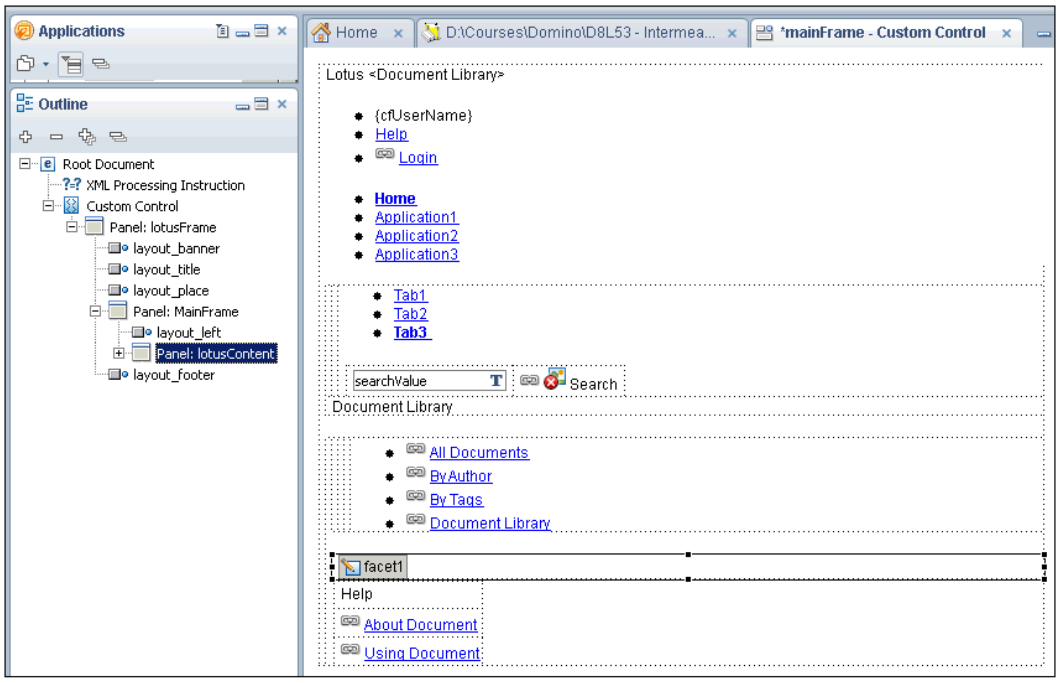
- Modularizes page structure.
- Provides more flexibility going forward.
- You will probably want to in the future anyway so easier to do it now.

The follow shows an example of the custom controls for the OneUI:

layout_banner	08/19/2010 10:...	Doctor Notes/WWCorp
layout_footer	08/10/2010 03:...	Doctor Notes/WWCorp
layout_left	08/18/2010 11:...	Doctor Notes/WWCorp
layout_place	08/10/2010 02:...	Doctor Notes/WWCorp
layout_right	07/27/2010 01:...	Doctor Notes/WWCorp
layout_title	09/01/2010 03:...	Doctor Notes/WWCorp
mainFrame	08/09/2010 11:...	Doctor Notes/WWCorp
title_search	09/01/2010 01:...	Doctor Notes/WWCorp

Do not forget to come up with a naming standard to help make your life easier. Here we use *layout_sectionname* for page section and *ccControlname* for content sections.

The mainframe control contains all the other layout controls, so it is what is placed on the XPage.



The panel named mainframe has the lotusMain style applied to it. All a developer has to do now is to create an XPage, drag the mainframe custom control on to it, and the custom control that contains the content they want.

5.2.3 CSS location

The CSS files you create can be placed either in the NSF or in the /data/domino/html folder on the server.
In folder:

- Available to all web application on the Domino server
- Not visible in design pane. Use dummy CSS to get around this.
- Can be automatically GZIP compacted - (server configuration).
- Can use dummy CSS to include in the NSF - ease of updates, see in design pane.
- Downloaded only once to the browsers.

In the NSF:

- Available only to current application.
- Visible in design pane.
- Easy to access and edit.
- Downloaded to the browser for each application it is used in,

Note: Take care to include only once on XPage (if on each control, it will be downloaded each time)

The CSS Frameworks used across application should be stored on Domino and a CSS that has changes and additions is stored in the NSF. This lets you make minor changes needed for the specific application when needed.

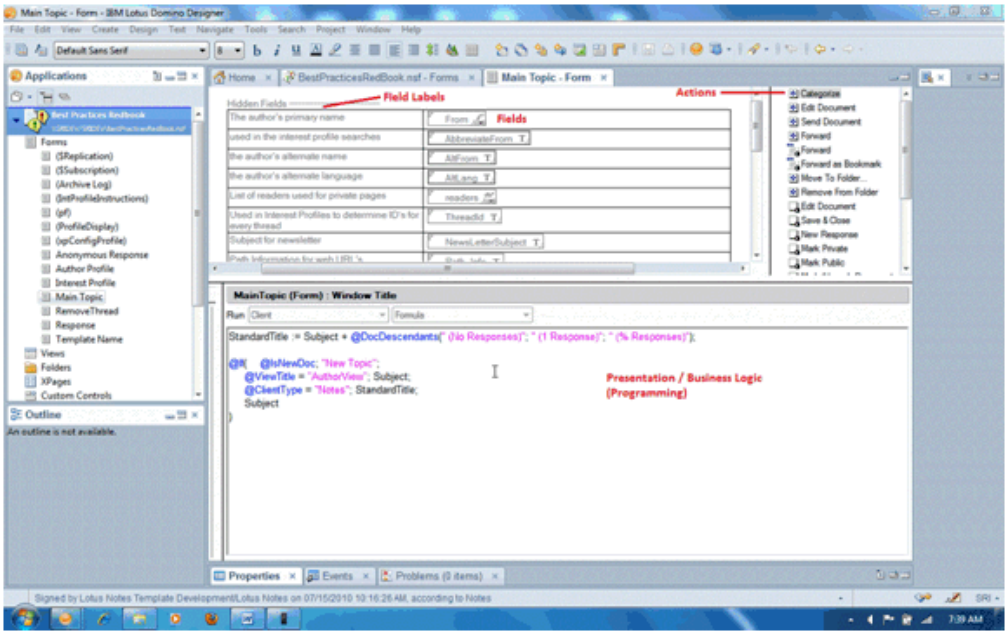
5.3. XPages versus forms

As application designers, you are challenged with finding ways to provide the application with rich, functional user interface components to accommodate the creation, modification, and presentation of data for Lotus Notes clients, web clients, and, mobile devices. This section reviews the *form*, a traditional design element for Domino applications, followed by the newest member of the Domino design elements: XPages. We discuss both and provide best practices in Domino application design along the way.

5.3.1 Domino form

In traditional Domino application development, a form is used for a variety of purposes, many of which we briefly review in this section.

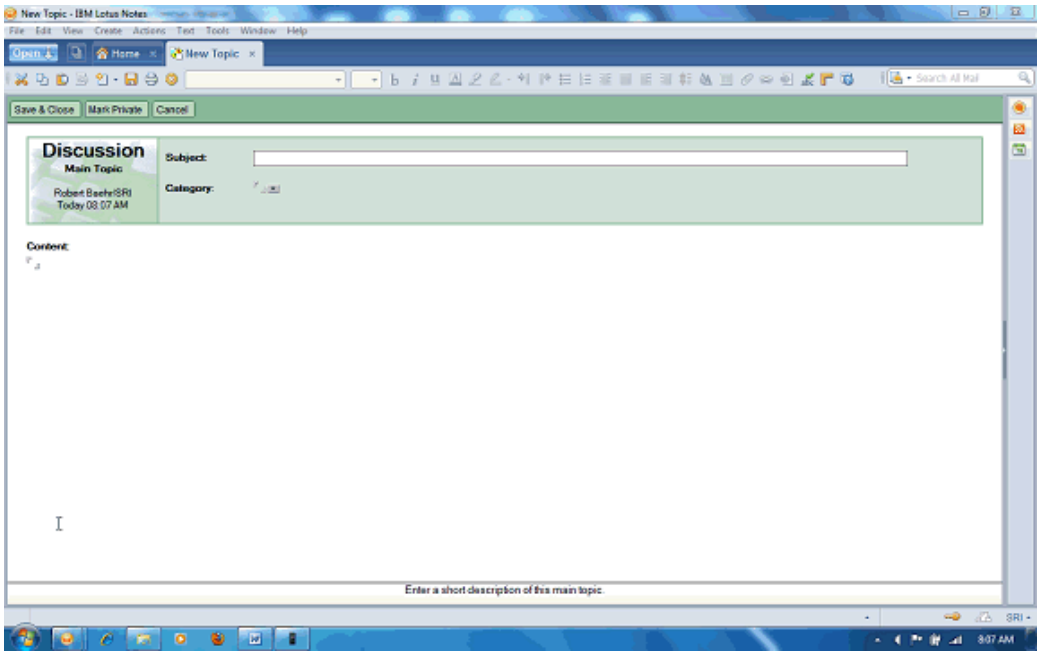
Metadata container
On a form, you can define fields which are used to hold a variety of data to be stored within the Domino application. The following figure shows a form example from the Domino 8.5.x Discussion Database:



The form contains fields, which house the metadata entered by the user, computed by Domino, or computed based on the results of an external process (depending on the design of the application).

User interface

Another important task that you must accomplish is to make the form functional and easy to use for the user. This is the basis of user interface design, as shown in a newly created document using the above form:



Notice that the metadata (Subject, Category, and Content fields) are bound to the form and its actions (Save and Close, Mark Private, or Cancel.), creating a *tightly-coupled* relationship between the form and document metadata.

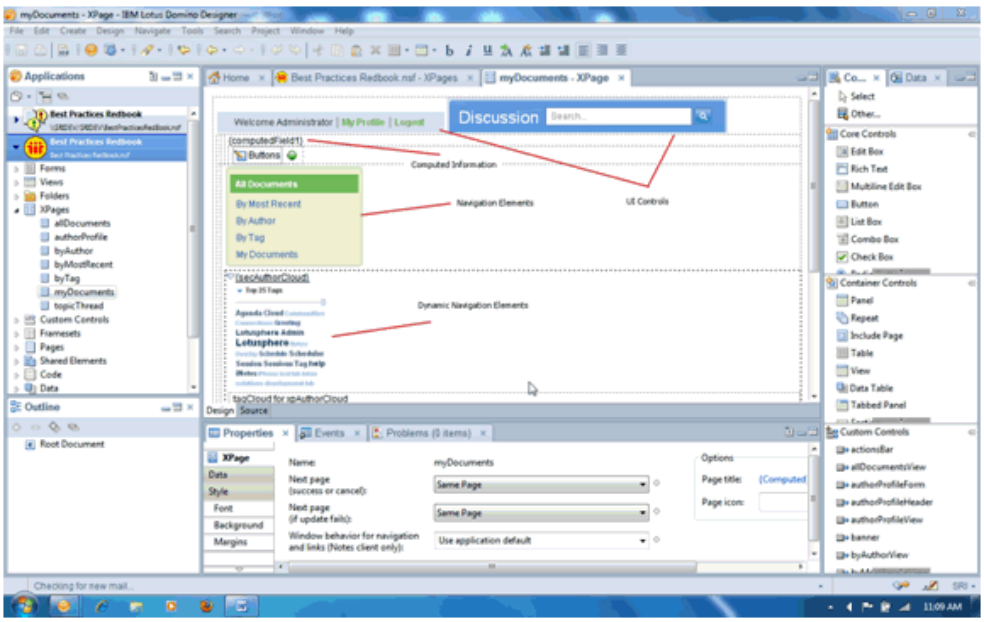
Business logic container
Also tightly coupled to the form and metadata is the business logic (custom programming). Business logic can include custom workflow processing, integration with external systems, and other computational functionality as needed. Since this is defined within the form (which contains the fields), there is an even greater tightly-coupled relationship between the document metadata, the form, and the business logic used to process it.

One of the biggest challenges of traditional Domino application design is to create a useable, functional interface for the Notes Client and web browsers. Traditionally, it is common to create two forms – one for use in the Notes Client and the other for use through web clients, each containing their own user interface components and associated business logic for the respective client. While this provides the desired results, we often find application maintenance a challenge, for you must implement each new feature twice.

5.3.2 XPages

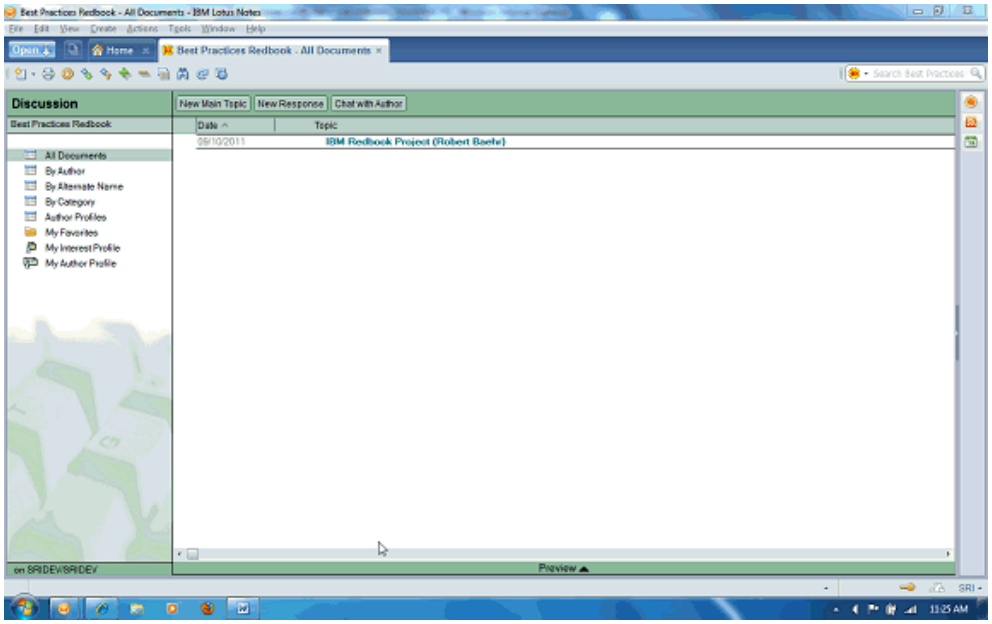
Introduced in Lotus Notes and Domino version 8.5.1, XPages offers a whole new way to design, develop, and maintain Domino applications for various clients (Lotus Notes, web browsers, and mobile devices). In effect, XPages gives a "write once, run anywhere" approach to application design. Here we explore some best practices for using XPages, including the integration with forms, server-side verses client-side processing, and rendering hide-when scenarios that, if implemented properly, will produce fast, efficient applications that are easy to maintain across various clients.

An XPage is a design element (similar to forms and views) that you can design to perform the same functions that traditional Domino design elements, for example, Pages, Forms, Views, Embedded Views, Navigators, Outlines, and more. As an example, the “myDocuments” XPage from the Domino 8.5.3 Discussion Template is shown below:

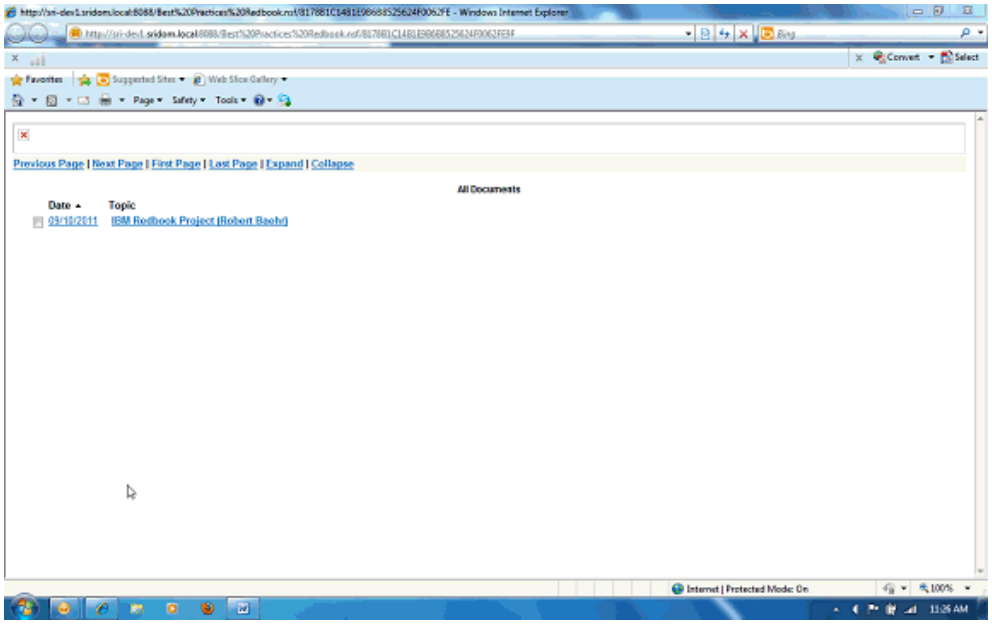


There are many different types of information that are present to the end user, all of which are contained in a single XPage. These items originate from the Controls List (along the right side of the screen), and are implemented in a drag-and-drop fashion. For each control, you can populate various code segments and properties to give the look, feel, and functionality that you desire.

While this might not seem like such an accomplishment, consider displaying the Discussion application using the traditional Domino view elements, as shown in the following two figures.
This figure shows a traditional view displayed in the Notes Client.



This is a traditional view displayed in a web browser.


Previous Page | Next Page | First Page | Last Page | Expand | Collapse
All Documents
Date
Topic
IBM Redbook Project (Robert Baehr)

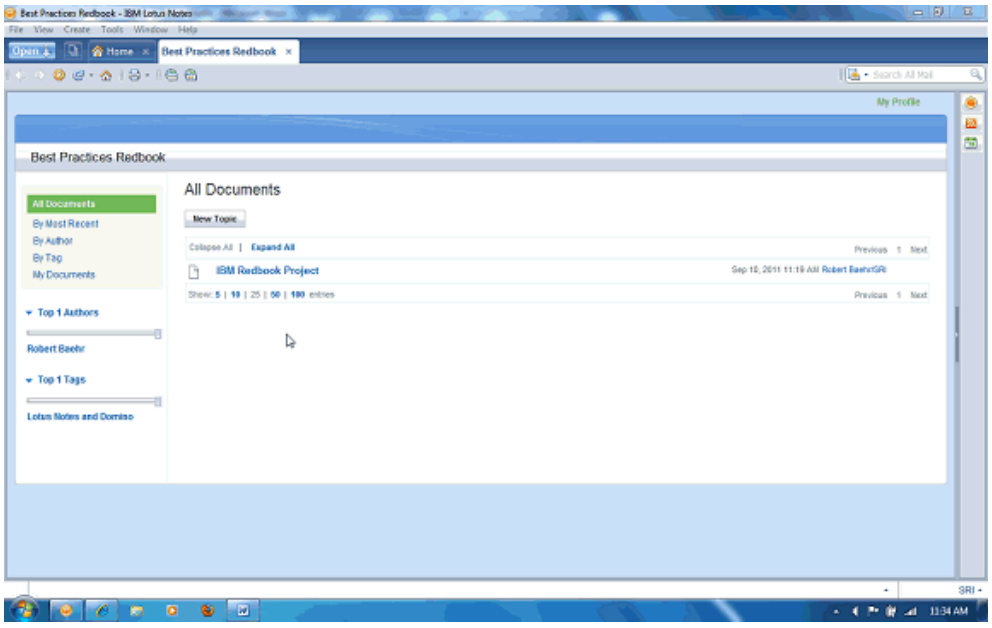
With respect to the above user interface displays, it is important to note the following:

- **Dissimilar user interface:** Note that, even though there is only one application, we must contend with two user interfaces – one for the Notes Client, and one for the web browser. This can increase user frustration which can impact application usability.
- **Navigation elements:** When two user interfaces are required, the developers have to define each piece of functionality twice – once for the web browser, and again for the Notes Client. Depending on the navigation element desired, this can greatly increase application development and maintenance time.

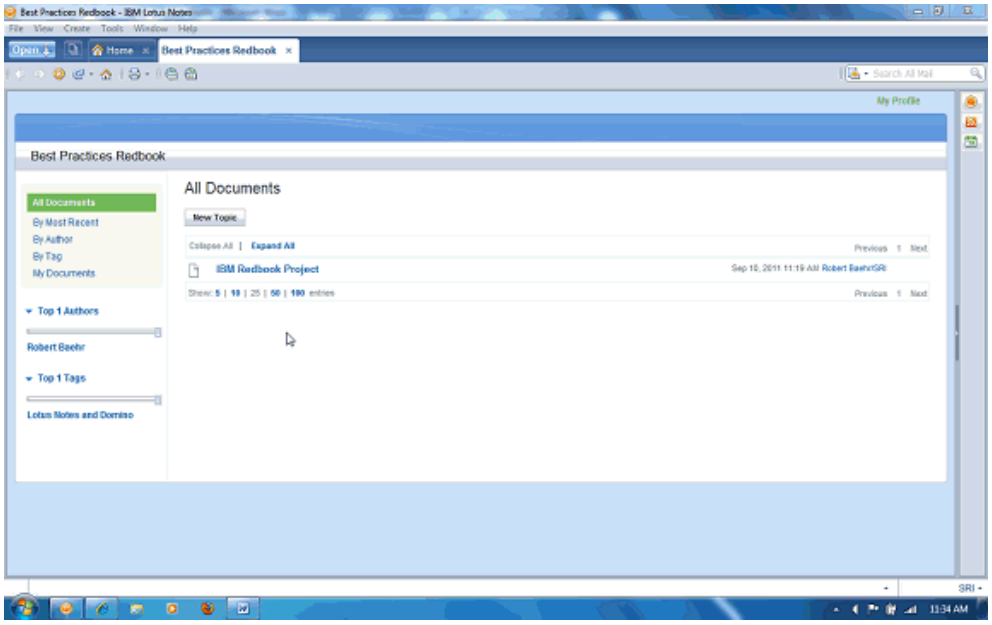
Other traditional Domino application design elements can provide the same level of dissimilarities in multiplatform development efforts.

The following two figures are user interfaces for Notes clients and web clients. These user interfaces have the exact same information but are with a single XPage-based design element.

This figure shows the XPages displaying a view In the Notes Client:



This figure shows the XPages displaying a view in a web browser:



The benefits of using XPages in the development of the user interface of a application should be obvious – you can create a single application that has a uniform interface and functionality, across the Notes Client and web browser platforms.

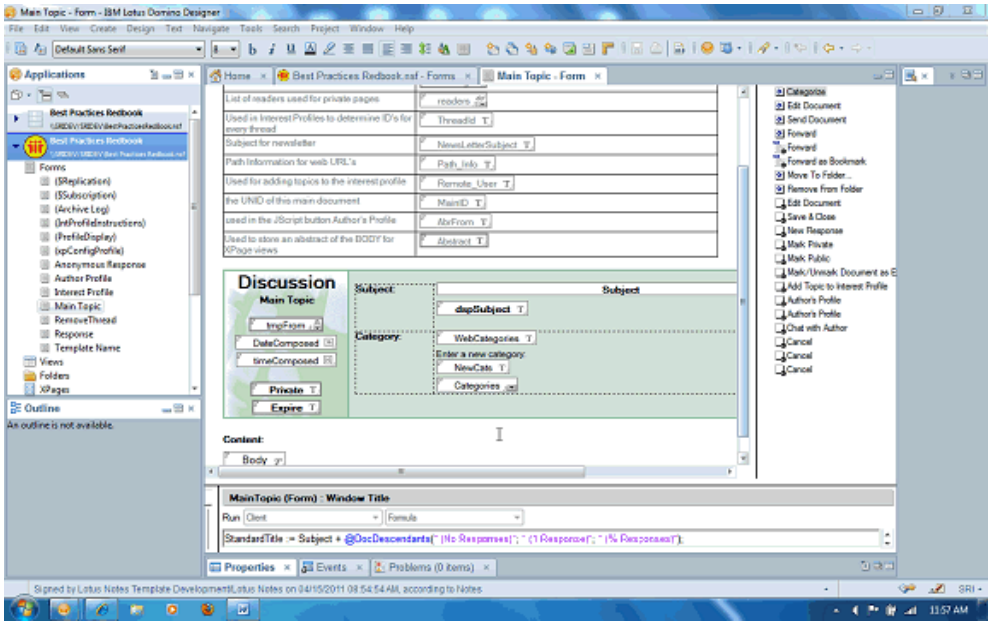
Best practice

From a fundamental application development viewpoint, it is advisable to fully utilize the XPages UI paradigm in application development.

5.3.3 XPages and forms

You can use a form to define the user interface, the metadata and integrated business logic for Domino applications, typically using one form per platform. Whereas, using the new XPages design methodology, you limit the use of forms to defining the metadata for a document only and use XPage to build the functionality, business logic, and user interface once. These are usable across multiple platforms.

As an example, consider the following Main Topic Form from the Discussion template for Lotus Notes 8.5.3:

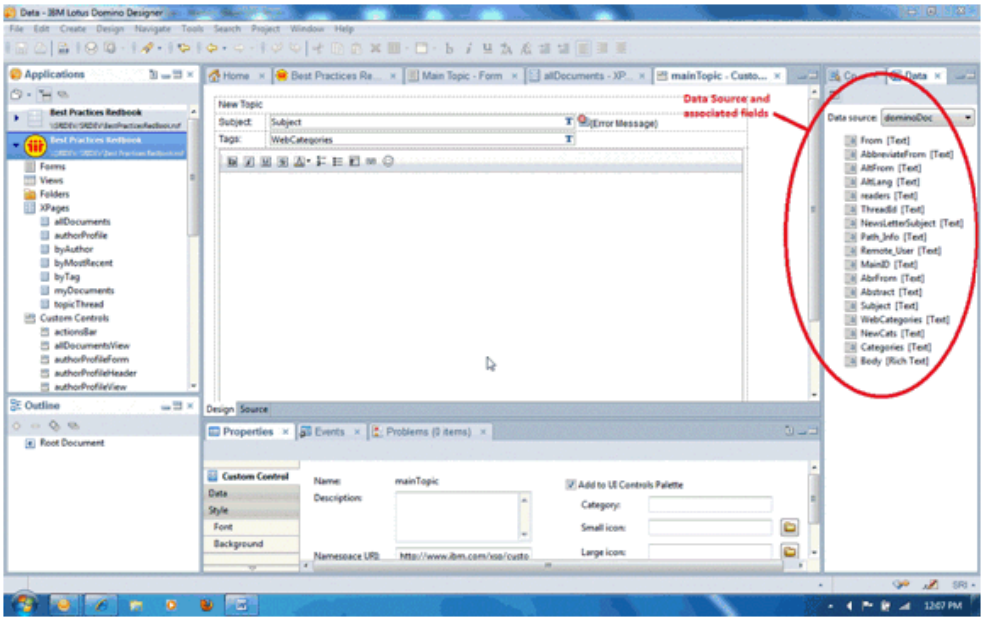


Here, the fields (the metadata), the actions (or business logic, along the right side of the diagram), and the user interface components (for example, the graphic in the background) are tightly bound to one form. Two forms are built, one for the Notes Client and one for the web browser.

Using XPages, you can use forms just to define the metadata, as outlined below:

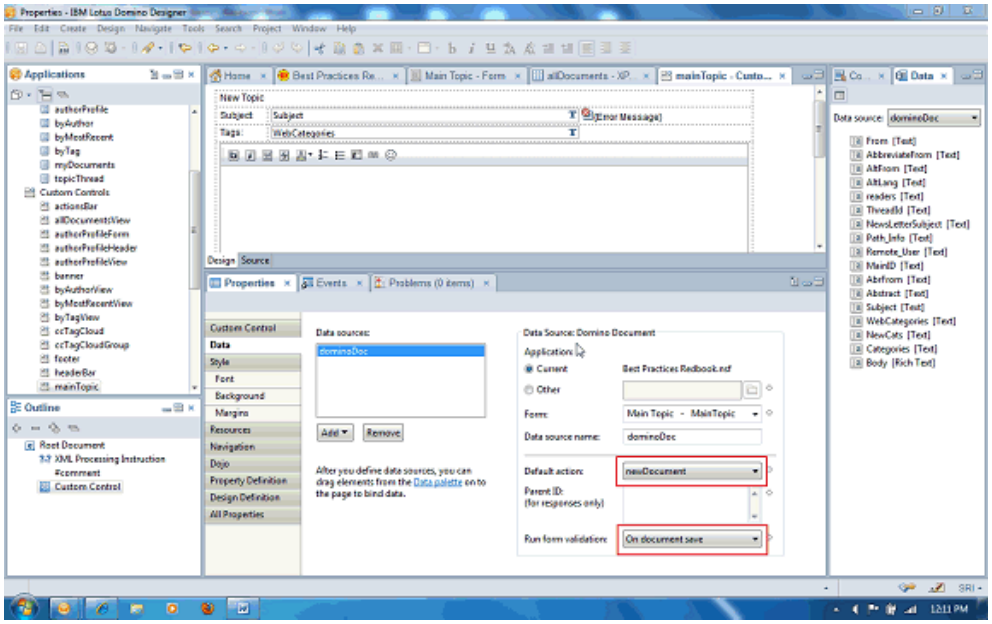
1. Define a form which contains the fields only (the field name, data type, and computed and editable properties and respective computed formulas). In XPages terminology, this form is referred to as a data source.
2. To design the actual user interface, define an XPage which uses the data source from step 1 . This makes the appropriate fields available to the user for data entry and manipulation.
3. The result is a single design element (the XPage) that can be used to create, edit, and delete documents using a consistent user interface and a single set of business logic.

To demonstrate, we use the Domino Discussion template for Notes 8.5.3. On this XPage, we find another XPages element, a Custom Control, as shown below:



The Data source field on the top of the right pan , shows that this XPage Custom Control is linked to the data source (that is form) called dominoDoc, allowing you to create documents using this form through an XPage (think of the XPage as a "front end" to the form).

In addition, you can place the metadata from the form on the XPage in a more "freely-flowing" format, giving you a user interface design that is flexible, functional, and modern. If we view the properties of the XPages Custom Control, you can see how this control and the data source are linked together:



A brief explanation is given below:

- **Default action:** The value newDocument instructs XPages to effectively create a new document, by default, whenever this Custom Control is presented to the user.
- **Run form validation:** If you have input translation and input validation formulas on you form, “on document save” instructs XPages to process those formulas whenever the document is saved. Although the preferred method of validation is to place the validation code on the XPage itself, Domino gives you the flexibility (which is especially useful during application migrations from traditional to XPages design methodologies).

5.3.4 XPages data validation and translation

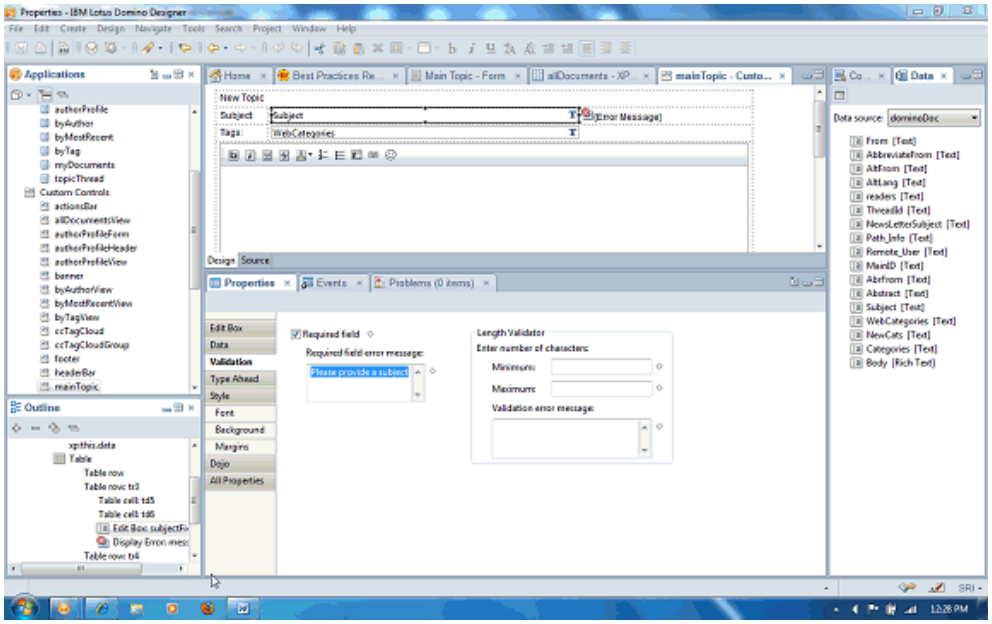
Validating the value in a field is a common programming technique and requirement. Assume your form required a a field called Subject that is a required field (in other words, you cannot save the document unless this field is populated). To accomplish this, you can write an input validation formula for the field, resembling the following:

```
@If(Subject = “” ; @Failure(“Please provide a subject”) ; @Success)
```

This approach has the following implications:

- From the Notes Client side, execution of this formula would be fast, for the formula is run on the Notes Client.
- From the web client side, because web browsers cannot execute @Functions, the execution of this formula would result in a round trip to the server for processing The browser would execute the validation formula, and, assuming that the Subject field is empty, return the result of the @Failure message.
- You could implement the input validation using Javascript, however, this would increase the complexity of the application.

With XPages, you can implement the same validation technique as shown in the following figure:



You simply highlight the Subject field, and, using the properties panel, to instruct the XPages that this field is required, and the message to display to the users if they do not enter data into the Subject field.

The result of using XPage validation is as follows:

- The validation properties execute identically on the Notes Client, web browsers, and mobile devices.
- The validation occurs on the client-side, maximizing application performance.

Note that the validation is not limited to the simple validation shown above. You can write full-scale business validation logic, instructing XPages to execute the code on the client side for maximum performance.

5.3.5 Visible versus hiding

With Domino application development, you have a number of tools which can change the user interface for the user at runtime – giving you power and flexibility over what the user “sees” and, how they ”see it.” This section review the techniques commonly used in traditional Domino applications (the hide-when) as well as the newer techniques brought by XPages (visibility and rendering).

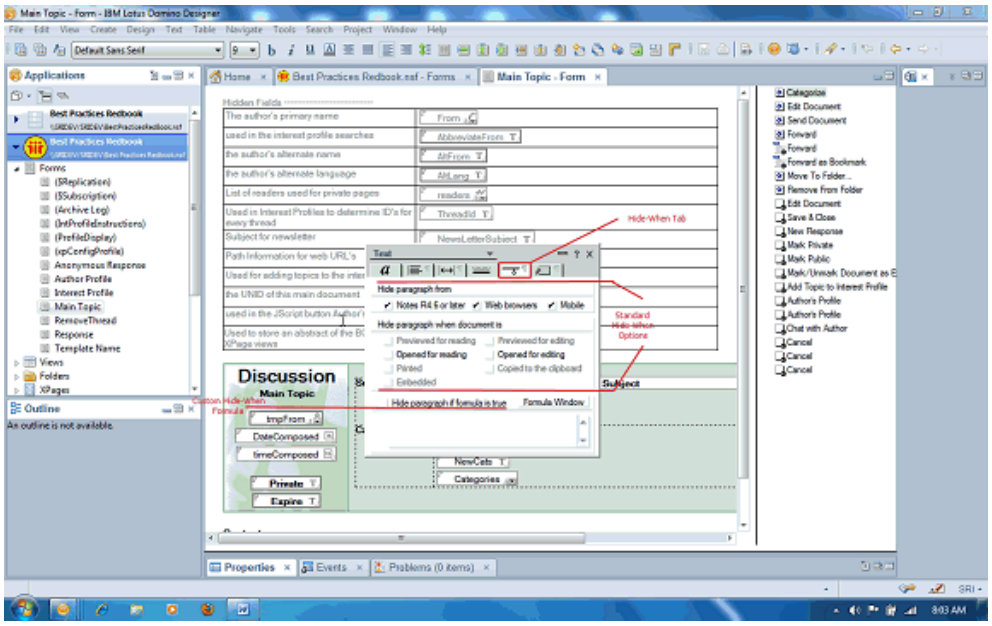
The hide-when options

Domino application development provides a mechanism that let you decide what user interface components (fields, text, buttons, actions, graphics, and so on) that a user is presented with when they open an application. These options, called hide-when options, allow you to quickly and easily hide various components from a user based on a host of criteria, including:

- When a user opens a document for reading
- When a user opens a document for editing
- When a user prints a document (Notes client only)
- When a user previews a document (Notes client only)
- When a user copies a document to the clipboard (Notes client only)
- When a user views a design element that is embedded onto another design element (for example, a view embedded on a form)

If these criteria are insufficient, you can also write a custom formula that must evaluate to true or false. In the event that the formula evaluates to true, the element will be hidden. If the formula evaluates to false, the element will be displayed.

As an example, see the anatomy of a hide-when properties box in the Main Topic form of the Domino Discussion template from Lotus Notes 8.5.3:



Domino automatically applies this formula based on "paragraph" boundaries, which include:

- Every item between the last time the user pressed the Enter key and the next occurrence of the Enter key
- The cell of a table

Try this
If you would like to fully understand the boundaries of the hide-when option, perform this experiment:

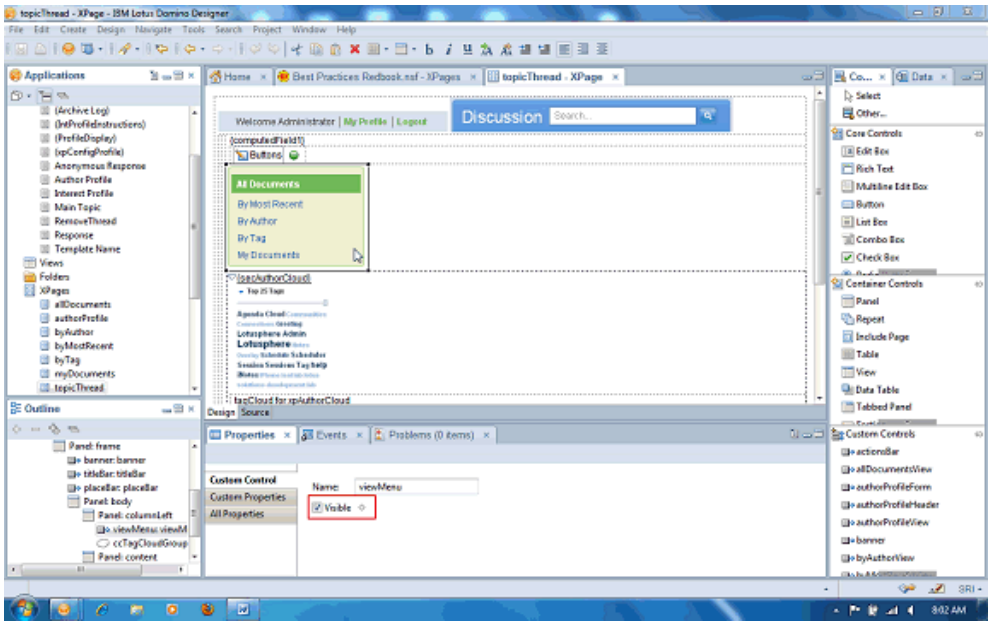
- In a test application, create a form named HWTest.
- On the form, press and hold the letter “A” key until you have multiple lines of the letter "A" on the form.
- Select one of the letters in the middle of the page, right-click to bring up the properties box, select the hide-when tab, and select the option "Hide When Editing".
- Preview the form

When you preview this form in the Notes Client (or a web browser), all of the letter A's are removed. Why? Because the hide-when formula was applied to the entire paragraph since we did not press the Enter key, the formula was applied to all of the letter A's back to the first occurrence of the letter A, and forward to the next occurrence of the letter A.

The visible property

XPages introduces a new property, the visible property, which allows you to choose whether you would like to view or hide an item at runtime.

Consider the following example from our sample application:



You have the navigation custom control selected for checking the Visible property. This instructs XPages to show the element on the client when you open this XPage. If you clear the checkbox, the element is sent to the browser, however, is hidden from us.

Note the diamond icon to the right of Visible in the graphic above. This is an indicator that allows you to write a custom formula to programmatically control the visibility of an element. If there is code behind the diamond, then there is custom code behind the object. If the diamond is clear, then there is no custom code behind the object.

Important note
To fully utilize the visible property, you must understand the difference between the hide-when and the visible property. In summary:

- The hide-when option instructs Domino to hide the element when the formula results to true.
- The visible property instructs Domino to show the element when the formula results to true, or the checkbox is checked, which automatically returns true.

Rendering

Rendering and visibility are often confused as being the same thing. While the end result is the same, there are some vast differences that you can encounter that can 'make or break' an application, including processing, delivery, and subsequent dynamic updates to the user interface.

Whereas visibility is generally done on the client, there are limitations that you will encounter when determining the visibility of an object, including:

- All information about the page is sent down to the client, regardless of whether it will be actually visible to the user. This can cause more data to traverse the wire than you actually need.
- Visibility is a fast way to dynamically update the user interface, for, since all data is in the browser, updates to the interface can occur quickly because processing occurs on the client.
- A limitation of the visible property is that all data required determining visibility must be pushed down to the client.

A situation might occur where the determination of visibility may reside in the value of data sitting on the server. Since the data is not on the client, you e cannot use that value to determine visibility.

Server-side rendering allows the Domino server to determine the visibility of an element, dynamically updating the user interface by passing data back and forth between the client and the server. The server, while processing an XPages component, can determine whether to send the actual component to the client.

The use of server-side rendering has its considerations, including:

- All processing of the user interface is done at the server, thus, more server resources might be consumed.
- Since the server must make the determination of whether an element is visible before sending data down to the client, performance of the application can be impacted.

As a best practice, it is always best to use client-side rendering when possible to assure that the application will scale and perform optimally.

Sharing design elements using custom controls

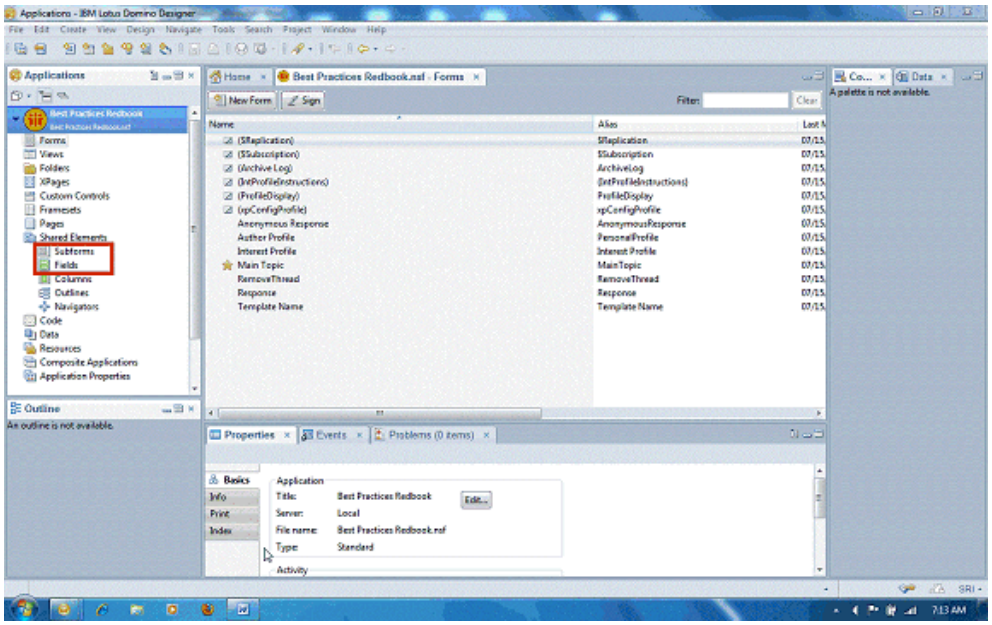
In traditional Domino application development, it is customary to design applications which promoted code re-use, componentizing the respective design elements for reusability. Some examples of traditional design elements which meet this endeavor include the subform, shared actions, shared fields, and more.

This section reviews traditional Domino design elements promoting code re-use, and, how they apply to the new XPages development paradigm.

Subform

Assume that, as part of your application design, you require the same fields on various forms through your application in the same layout (that is. user interface), translation and validation formulas (among others). Traditionally, you could use the Subforms and Field to accomplish this goal by

- Placing each field on each form, independent from one another (or, use shared fields so that we only have to define the field one time).
- Placing the fields on a subform, then insert the subform into the main form as needed.



Because the XPages do not support integration with subforms, the above approach is a challenge in the application design.

Best practice
When designing a new application, or, modernizing an application to use XPages, remember that direct subform integration, as well as other traditional Domino shared items, are not supported. For more information, see Domino Designer help for information related to traditional Domino design element integration with XPages.

Custom controls
A custom control is a reusable design element used in XPages that can contains various pieces of information, including, but not limited to:

- User interface components
- Fields
- Views
- Business logic
- Static information
- Dynamic information
- Graphics

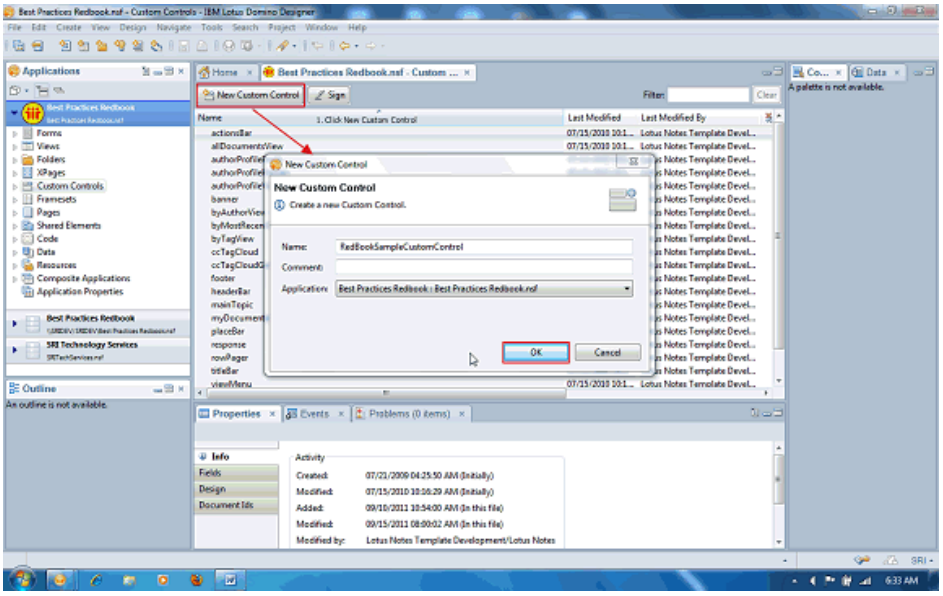
Similar to subforms, you create a custom control in Domino Designer, and insert the control onto one or more XPages. Thus, modifying the custom control automatically update each of the XPages that use it (promoting code re-use).

One example of how custom controls can help you build powerful applications is the integration of XPages applications with OneUI, a series of custom controls that permit

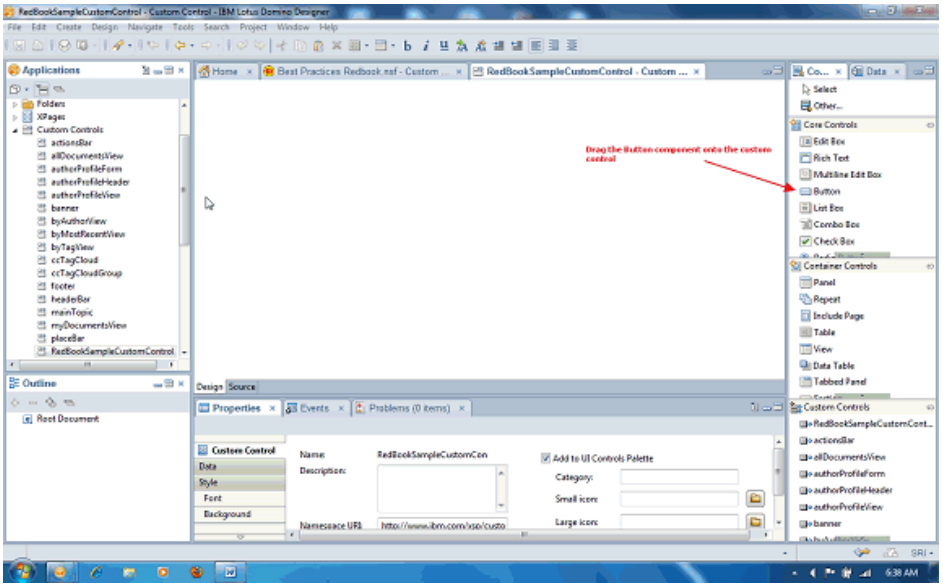
you to develop consistent, reusable standard interface objects, navigation object, and more. For information about OneUI, see: http://www-12.lotus.com/ldd/doc/oneuidoc/docpublic/index.htm

Creating a simple custom control
Use these steos to create a new custom control in Domino Designer:

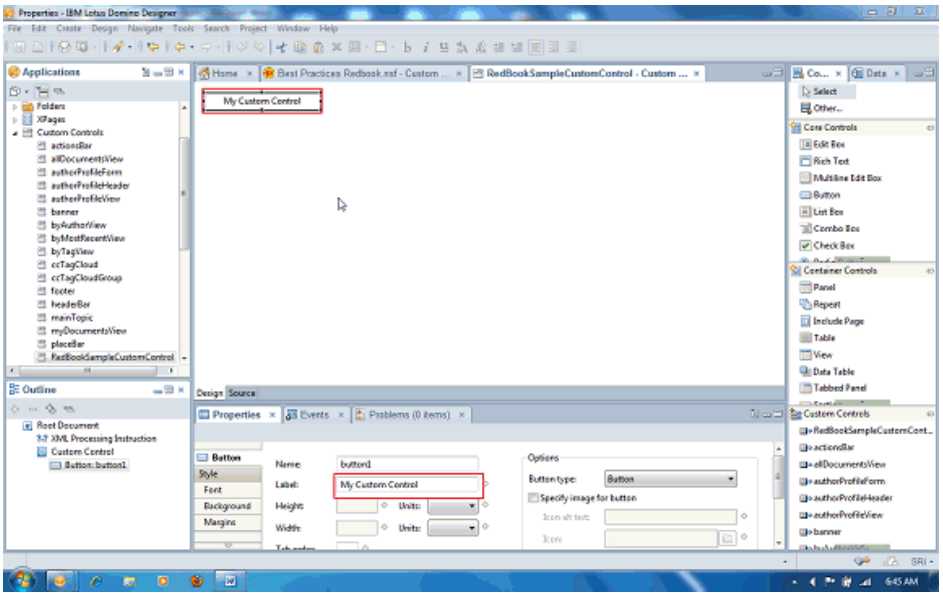
1. Click **New Custom Control**, enter a new for this new custom control and select the application. See the figure below:



2. After clicking **OK**, you are placed into the custom control editor in Domino Designer. Drag the button component from the controls list onto the new custom control, as shown below.

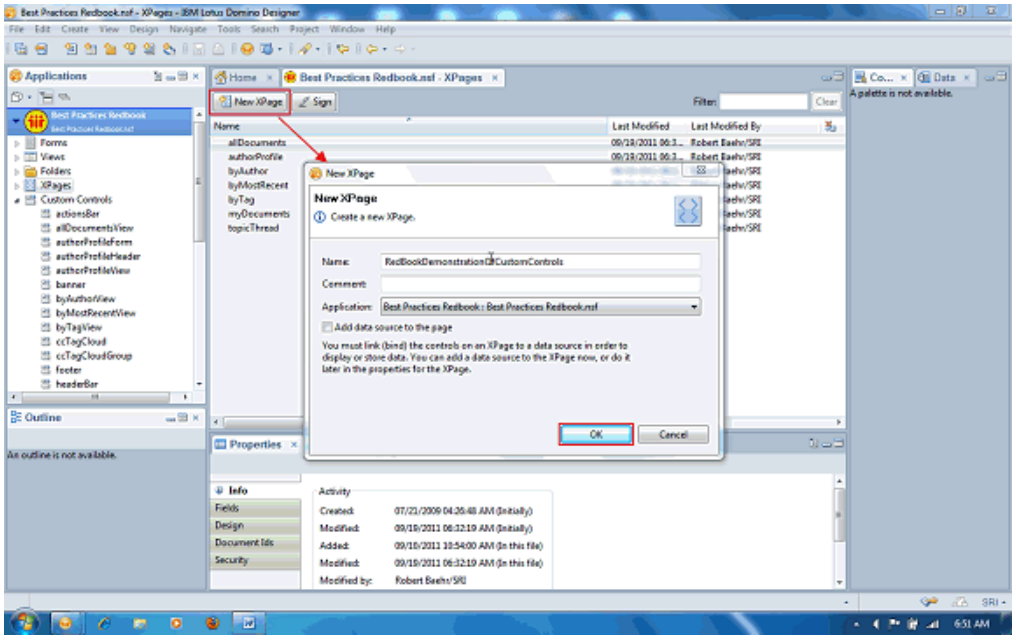


3. The result is a single button whose properties you can modify to control the look, feel, and functionality of the button. In this example, we change the Label property to "My Custom Control", which results in the following:

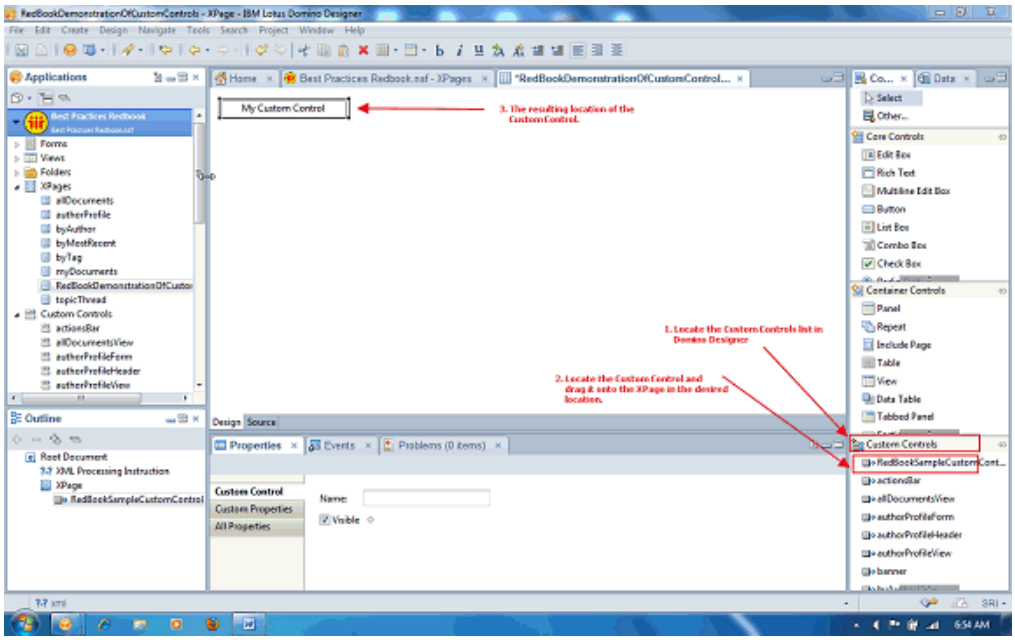


4. Save and close the custom control.

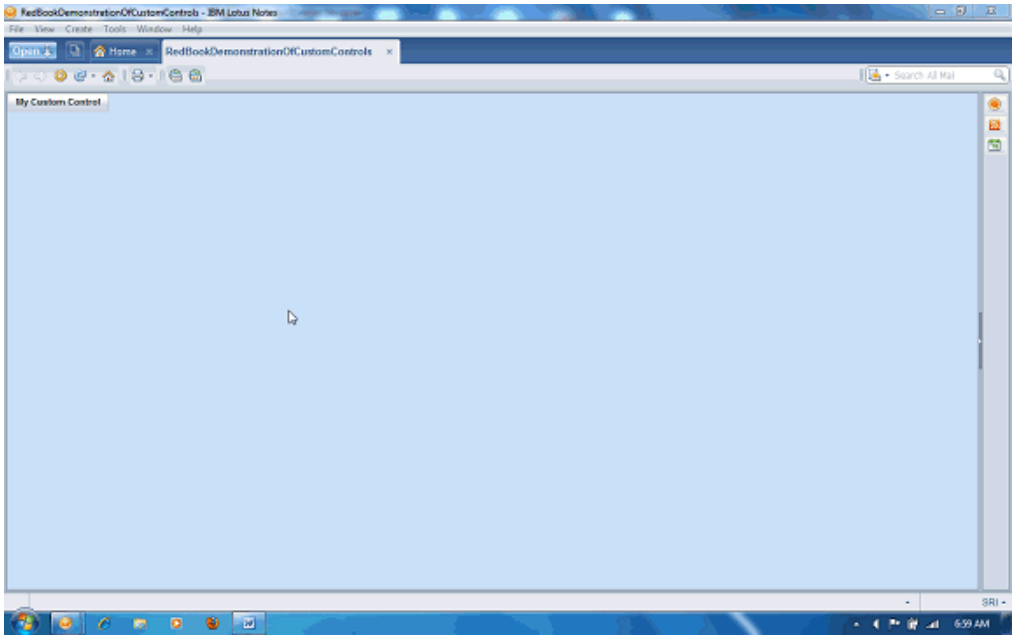
To reuse the custom control, create a new XPage in Domino Designer, as shown:



You can easily see all custom controls in Domino Designer by looking at the Custom Controls list in the lower right section of your screen. To use the custom control on the newly created XPage, you simply drag and drop it to the desired location, as shown:


My Custom Control
3. The resulting location of the Custom Control.
1. Locate the Custom Controls list in Domino Designer
2. Locate the Custom Control and drag it onto the XPage in the desired location.

Upon preview in Lotus Notes, you see the following XPage with your newly created custom control.



Note that the above XPage appears virtually identical in other client environments, such as web browsers.

Assuming that you use your custom control in multiple XPages, if you change the custom control, you automatically propagate that change to all XPages that it is used on, promoting element and code reuse.

You can now repeat the process of re-using the custom control on a variety of XPages, taking advantages of user interface design and code reuse in your Domino applications.

5.3.6 Best practices

Domino provides you with a host of application design features. When developing a Domino application, consider the following:

- Create forms to define the metadata for documents, and integrate associated Lotus script agents in the appropriate events (QueryOpen, QuerySave, and son on).
- Use XPages to develop the actual user interface, using the respective form(s) as a data source for the XPage.
- Use XPages to develop applications to write applications and support multiple platforms consistently.
- Carefully consider the use of server-side versus client-side data validation to prevent round-trips to the server for data validation and translation. Use client-side validation and processing when possible.
- Carefully consider the use of the visible property versus rendering to assure that the information that the user sees, as well as the application functionality, is processed correctly.
- Use custom controls over traditional Domino shared design elements (such as subforms, shared actions, and shared fields) to create reusable user interface components, business processing logic, and other reusable elements in your application development efforts.

5.4. Accessibility

Accessibility is a general term describing to which degree an application is available to its users. It can be understood as the ability to access an application with. While accessibility in the past was focussing on people with disabilities or special needs, the current understanding has broadened within the IT industry to focus on a wide range of different devices with different capabilities each.

The broad definition of accessibility means applications are accessible to a greater range of devices and applications. These devices and applications can reinterpret compliant information for a broader audience, for example, into voice by page readers or reuse through mash ups.

5.4.1 Why to think about accessibility

Your business should think about accessibility for the following reasons:

- Avoid legal disputes
  While many countries have defined accessibility requirements by law (See examples below), those definitions might not be clearly defined within the requirements provided in the beginning of your application development project. Not knowing about these definitions does not prevent from becoming the target of legal action against you or your company for not ensuring your application is accessible.
- Competitive advantage
  When building a Domino application, you are most likely building it for a defined

audience, a certain type of user or device type. However, think about the audience you can reach out to if your application would support a broader range of devices or languages. Especially with mobile devices being a major client type your application will gain more interest if its easy to access on all different clients and device types.
- Enhance your corporate image
  Negative reports have been in the press about companies who have not tried to make their products and services accessible to everyone.
  Taking positive action should have the reverse effect and enhance your company image in the marketplace.

The following examples of legislation have provisions requiring physical accessibility:

- In the U.S., under the Americans with Disabilities Act of 1990
- In Australia, Disability Discrimination Act 1992
- In the U.K., the Disability Discrimination Act 1995
- In Germany, the "Bestimmungen für eine barrierefreie Informationstechnik" § 11 BGG

There probably are many more. Check with your legal team to find out what regulations do apply to your project.

5.4.2 Considerations

While the functionality aspect is often described well, developers are requested to implement those definitions by using the platform of their choice: Domino. Notable challenges are:

- There are legitimate reasons to switch JavaScript support off.
  For example, corporate security restrictions filter JavaScript already at firewall level, or the Firefox plugin "NoScript" is in use to protect people from cross-site scripting (XSS) or Cross-Site Request Forgery (CSRF) vulnerability.
- Many devices ignore JavaScript entirely.
- Screen readers do execute JavaScript, however, you may not want them to.

5.4.3 Approach

Building applications for accessibility requires a little change in mindset. While some issues are quickly resolvable by adding some client side JavaScript, it may be difficult when thinking about a broader range of devices. So one approach is to think in layers, a basic functional layer, a CSS (for example, OneUI) style sheet layer to apply a visual design, and some JavaScript on top to enhance the site's behavior.

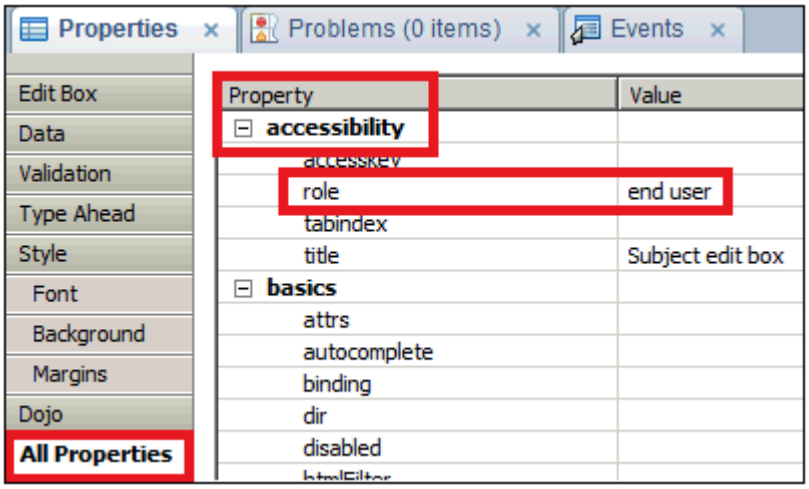
While there is a complete documentation available, a little explanation about the how and why goes a long way. The OneUI is not so much about colors and fonts, but about structure, navigation, and layout. Following its guidelines, your applications get a consistent look, which user studies show increase user confidence about their ability to use your web application. This translates into less help desk calls. Any consistent framework does that, the advantage of the OneUI is that your applications start looking consistent to the IBM software in use. Furthermore, when IBM releases a new visual style, your application can adopt it in no time.

One remarkable new feature of Notes/Domino 8.5.2 and beyond is the consequent use of Aria tags for accessibility.

Aria tags allow specific devices to identify the purpose of an element on the page, literally, the role attribute provides information on what the object is.

5.4.4 Usage of Aria tags in XPages

Accessibility tags are defined within the properties of a control. To define them, open your application in the Domino Designer, edit the XPage or the custom control design element, click **All Properties**, and look for role under accessibility.



This property implements the W3C ARIA specification on roles that are described further in http://www.w3.org/TR/wai-aria/#Using_intro.
Screen readers such as JAWS use roles to acquire contextual information about HTML pages.
For valid values, see http://www.w3.org/TR/wai-aria/#roles and http://www.w3.org/TR/wai-aria/appendices#quickref

5.4.5 Example

This Edit Box control provides a title and role for accessibility.

```
<xp:inputText id="inputText1" value="#{document1.subject}"
        title="Subject edit box" role="end user">
/xp:inputText>
```

5.4.6 What needs attention

Building accessibility can be a broad range. However, it helps to keep a few items in mind that you can apply:

- Have you checked the sequence of your Tabs?

```
tabindex="x"
```

- How about keyboard shortcuts? Did you assign them everywhere? do they work as expected?
- What about navigation shortcuts for screen readers?

```
<span class="show_screenreader_only><a href="#nav" title="Main
Menu">Main Menu</a></span>
```

- Contrast between foreground and background, custom style sheets might look smart but do not allow people with disabilities to read foreground text on a background with a similar color.
- The colors, "simple" text, alt-text for Images, and so on used in the user interface elements should be available for all display types.

While this list is not aiming to be complete checklist, it gives a certain impression what requires special attention when building an application.

5.4.7 Useful tools and resources

This section provides a list of useful tools for developing applications for accessibility.

- IBM Human Ability and Accessibility Center - Web checklist
  http://www-03.ibm.com/able/guidelines/web/accessweb.html
- Firefox Web Developer Plugin
  https://addons.mozilla.org/de/firefox/addon/web-developer/
- Internet Explorer Developer Tools (hit F12 key)
- Lynx - a text web browser
  http://lynx.isc.org/
- NonVisual Desktop Access (NVDA)
  http://www.nvda-project.org/
- Coblis - Color Blindness simulator
  http://www.colblindor.com/coblis-color-blindness-simulator/
- Colour Contrast Analyser Firefox Extension
  http://juicystudio.com/article/colour-contrast-analyser-firefox-extension.php
- Contrast Analyser
  http://www.paciellogroup.com/resources/contrast-analyser.html

5.4.8 Conclusion

Adding accessibility features into an application is not hard, but it is also never finished. New regulations and standards appear faster than you expect them.
Working with local accessibility group will certainly help developers who are new to this subject to understand the challenges people can have.

5.4.9 Further references

- W3C Candidate Recommendation - Accessible Rich Internet Applications (WAI-ARIA) 1.0
  http://www.w3.org/TR/wai-aria/#Using_intro

- Dijit Accessibility (a11y)
  http://docs.dojocampus.org/dijit/a11y/index?action=show&redirect=dijit/a11y

5.5. Graphics

The main best practices consideration as regards graphics is that they may be overly large or you might have too many of them, resulting in poor performance. But it also makes sense to pay attention to the quality of the images themselves.

5.5.1 In General

Avoid having multiple copies of the same image; use an image resource design element or other common resource area so that the user's browser or Notes client can take maximum advantage of caching.

Try to use PNG files wherever possible. Modern browsers now offer full support to PNG format which not only supports most of the palette options offered by GIF but is also much smaller in size. In addition, PNG graphics support use of an "alpha layer," by which you can make pixels partially transparent. This lets you use the same image on different backgrounds without worrying about edge effects where the border of your graphic is jagged, or where there's a "halo" of wrong-color edge pixels.

Always use 100% size of your image. There is no need to scale your images because even if you do, the browser or client will still download the full image. Therefore, in the end, your code ends up doing more work rather then less.

NOTE: Because there is such a wide variety of screen sizes and resolutions these days, in some cases it might make sense to scale graphics down somewhat so that they can be zoomed in a little without getting jagged or blurry. But be sensitive to the fact that the browser/client will cache the entire image, and don't use a higher resolution than you actually need. Don't for instance, make a thumbnail by scaling down a full-screen image. Have a separate thumbnail file.

Use freely available tools to optimize and analyze image palettes and size, and try and filter out the unwanted components. This will help you, especially, in low-bandwidth environments. Some of the tools that you can use are *imagemagick*, *jpegtran*, and *pngcrush*.

5.5.2 In the Notes Client

When placing graphics on forms and subforms, use image resources rather than pasted or imported graphics, if there's any chance that the image might be reused.

If not using an image resource, use imported graphics in preference to copy/paste. Pasting uses a proprietary Notes bitmap format encoding, which is not very efficient. Importing preserves the original format, so that your efforts to minimize image file size beforehand aren't wasted.

5.5.3 In Web Applications

For graphics-heavy applications, consider using image sprites which contain a collection of several images combined into a single image file. You can then use these sprites through a combination of tags and CSS properties.

Move your more often used resources to the server's **html** directory. Loading your files from the server file system is faster then loading from the individual database, and takes maximum advantage of browser caching (since all the applications on the server can share those files). You also have the option of compressing your files using GZip and then upload them to the server

5.6. Design for target audiences

When building an application, developers have to resolve more than just a business problem. Today's applications are requested to function on more than just one type of computer, one type of operating system, browser, or device. The way people work at their desktop computers is completely different from the way they work on mobile devices. Each device type has its own benefits and limitations; however, you want to provide your application to a broad range of user types.

While classic Notes applications are only used by a Notes client, the modern applications have to serve a variety of client types. Often, the type of client is not even defined when the application design starts. However, development teams should give thoughts to the audience for whom they are trying to provide the application.

When design your Domino applications, always design with the audience in mind and consider the who, what, where, where, why, and how.

5.6.1 Who is the target audience?

Does your target audience consist of the IT Department with a small set of of mobile devices? Is it the Executive Board who often delegates tasks to their personal assistants, or are they comprised of warehouse staff working from kiosks? Knowing who your users are will greatly direct the project (and as a result the end-result) to the right direction. It is extremely critical that you understand both who they are as well as their day-to-day interaction with the organization.

We recommend creating role-based user profiles -- written bios, not to be confused with the user roles or profile documents in the IBM Lotus Notes Domino Application Development -- for each of the potential user types. Submit those description for review to your project champion.

With these profiles in place, you not only confirm your understanding of your solution user base, but also proactively address any scope creep specific to who will use the application.

5.6.2 What is the goal?

Once the audiences of the application are clarified, find out the goal of the sponsor of the project. A goal should be described clearly. Too brief of a goal description, such as "mobilize a given Notes database" is not adequate and can lead many open questions in application functions required. The goal here should be to address the stakeholders' goals while ensuring that a user experience that lends itself to the target platforms is created.

Take an example, a personal address book focuses on the contact management features only but left out the other functions such as connection documents, locations, accounts, and so on might be an application with a Contact form that has over 33 input fields enabled by default. When sizing that down to a mobile device, the users will not be happy with the result. Usually, someone accessing a group Personal Address Book using their iPhone or BlackBerry would not care about a given contact's second personal email address.

There will be compromise, and the mobile user experience, when done right, will not simply be a port of the rich client Lotus Notes or web browser version of the solution to a 320x480 resolution screened mobile device.

5.6.3 Where will they use it?

The ideal is "Everywhere". However, this is not an easy task. Consider the following:
Not everyone has decent mobile device coverage. Think about the users who has the type of reception that could only result from lead-lined walls built for anti-kryptonian measures. WiFi can also be problematic. Your high speed internet access solution can fall flat if people receive connectivity errors when they click on that new icon of your application on their mobile devices. User perception is reality, and errors tell them simply that it does not work.
Will you need offline or local storage capabilities? This depends on connectivity and coverage considerations and just how critical of a solution you are requested to develop. Is relying on a local data store that is potentially several minutes out of sync with the master database an issue? Does it pull down system-generated unique and numerically-sequenced IDs? Does it need to actively communicate to live systems to function? The answer highly depends on the type of application you are focussing on.

Once you have answered these questions, or at least brought them to the attention of the project champion who might not have considered that the warehouse staff could go "offline" when checking for real-time stocked inventory numbers for product substitutions. Can you grasp the architectural considerations required to even make this solution work?

5.6.4 When do they need it?

Understanding the delivery expectation for your application, and communicating about the solution performance potential (whether that is shorter or longer than the project champion assumes) will help everyone involved. It is about establishing a realist delivery expectation by breaking the expectations into these categories:

- Need to have
- Like to have
- Nice to have

Make sure that the "Need to have" category addresses the business critical goals that you previously established. You can set the time line for delivering the items in the "Like to have" categories based on the available resources. The "Nice to have" items receive the lowest priority.

5.6.5 Why would your users use it?

When answering the "why", the major consideration is "Does the individual user receive a value from using the application?"

If the answer is no -- if the user has to do more work, if the process is now more complicated, or if this entire exercise is simply because someone told to do so, then the project is highly possible to be one of the ill-fated projects. Should this be the case, think seriously about innovation that helps the user to gain some value out of it; or if even this is not possible: question the project existence.

5.6.6 How can it be done?

The final question of "How" is the result of the developers' knowledge of the platform -- using XPages technology.

Chapter 6. Performance considerations

6.1. View index performance considerations

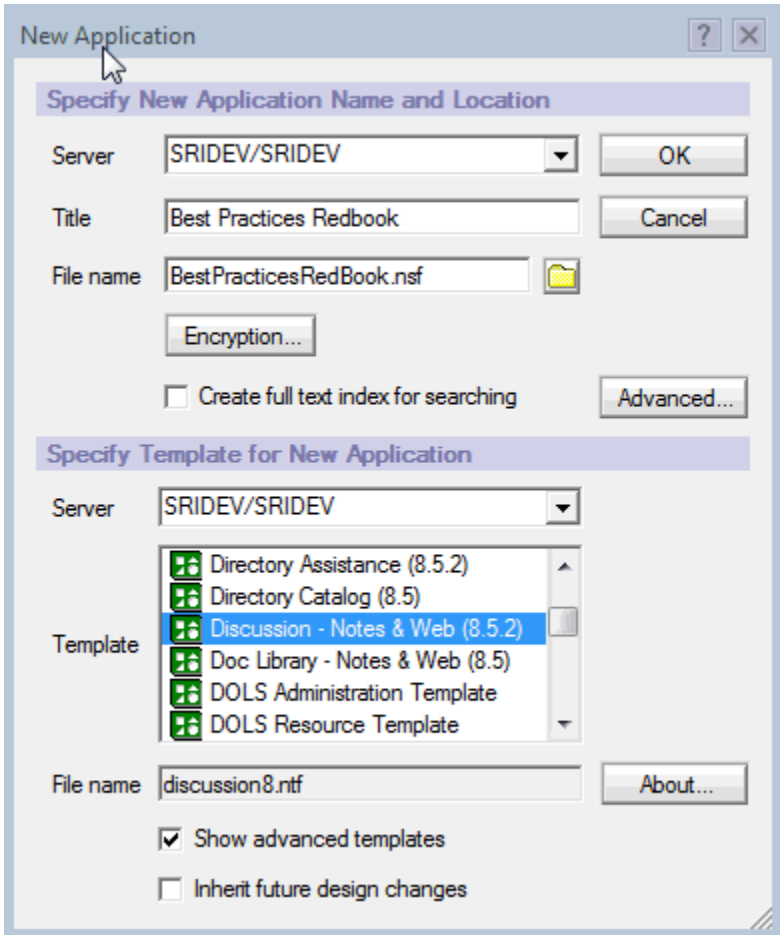
6.1.1 Introduction

Views are the primary consideration in performance. When you're having a performance issue, views are generally involved somehow.

Most views are indexed automatically by the server, so that they can open quickly when users want to use them. This means that a poorly designed view will often not be noticeably slower to open than other views; but it will drag down the performance of the server and make the server fall behind in its view indexing task, so that all views in all applications on the server experience a slowdown.

To demonstrate the different techniques of efficient view design and use, we use a test environment and sample application. For this section (and other sections, as noted), we have the following environment:

- **Server:**
  - Domino Server Version 8.5.2 (with FixPack 2 applied)
  - Operating System: Windows Server 2008 Enterprise Edition (running in a Hyper-V environment
  - Memory: 4GB RAM
  - CPU: Intel Zeon Processor (3.33GHz)
  - Disk: Three disk drives, expanded as follows:
    - C: Drive: Operating system
    - D: Drive: Lotus Domino 8.5.2
    - E: Drive: Domino Transaction Logs (more on this topic in section 6.1.4)
- **Client:**
  - Lotus Notes and Domino Designer 8.5.2 (with the FixPack 2 applied)
  - Operating System: Windows 7
  - Memory: 4GB RAM
  - CPU: Intel Core-Duo (1.8GHz)
  - Disk: One C: drive to accommodate the Operating System, and Lotus Notes and Domino Designer 8.5.2
- **Application:**
  For the purposes of illustration, we use the newly-revised Domino Discussion template that is installed with the Domino server, as created below:



All references to this section (and others, as noted) are made to the same application, server, and data set for consistent findings. We create 10,000 documents in the application (using a LotusScript agent) to show the performance metrics.

Factors impacting view indexes

From an application development perspective, there are many design decision that we make every day that impact view indexing. Depending on the decisions made, view indexing (thus, view performance) can be effected. Here, we discuss how the view indexes are maintained and their performance implication when these design decisions are implemented:

- Column sorting and categorization
- Using @Functions, and especially, date/time based functions
- Transaction logging
- Controlling Domino view index maintenance
- Using XPages

Now that we have our environment established, let us dive into the topic of one of the most frequently ignored facets of Domino development: the View index.

6.1.2 View index sorting considerations

Each view in a Domino application has an index, which is essentially a mini table of information that Domino maintains and uses to determine:

- What documents to display in a view
- What order the documents appear in
- What other column information is displayed in a view

It is our goal to design applications to keep this index as small as possible, for, the larger the index, the more work that Domino has to do – hence, the slower the view response time will be for the user. How the views are sorted impacts the size of the index. You have the following view sorting options:

- No sorting
- Single column sorting
- Multi-column sorting
- Dynamic (user-controlled) sorting

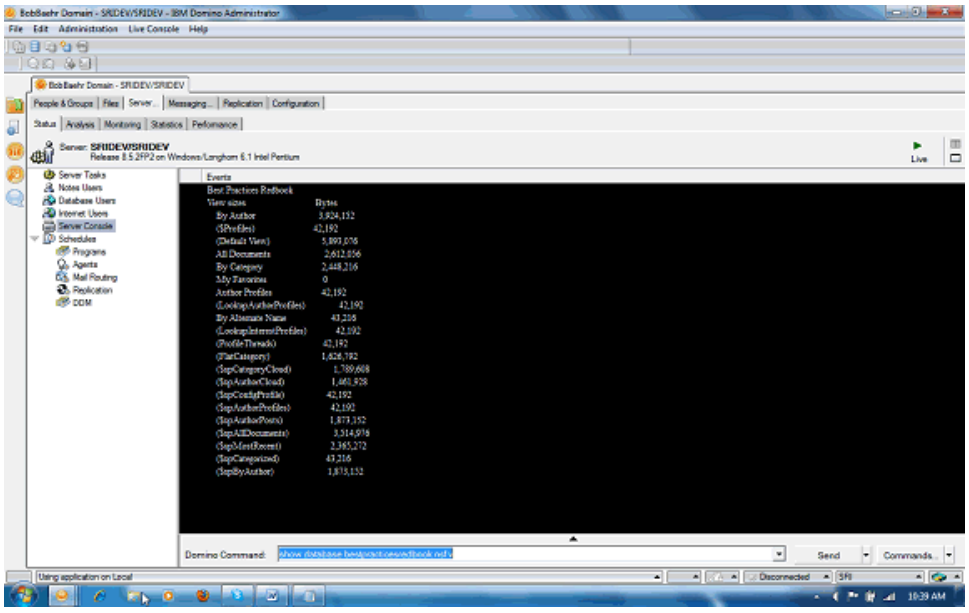
You can display a list of view and the size of each view index using the following command:

```
SHOW DATABASE dataBaseName.nsf v
```

Here we show the view index size under each sorting options in the sample application.

No column sorting

The follow figure shows the output of the SHOW DATABASE command on a sample "All Documents" view which has no sorting applied to any column:



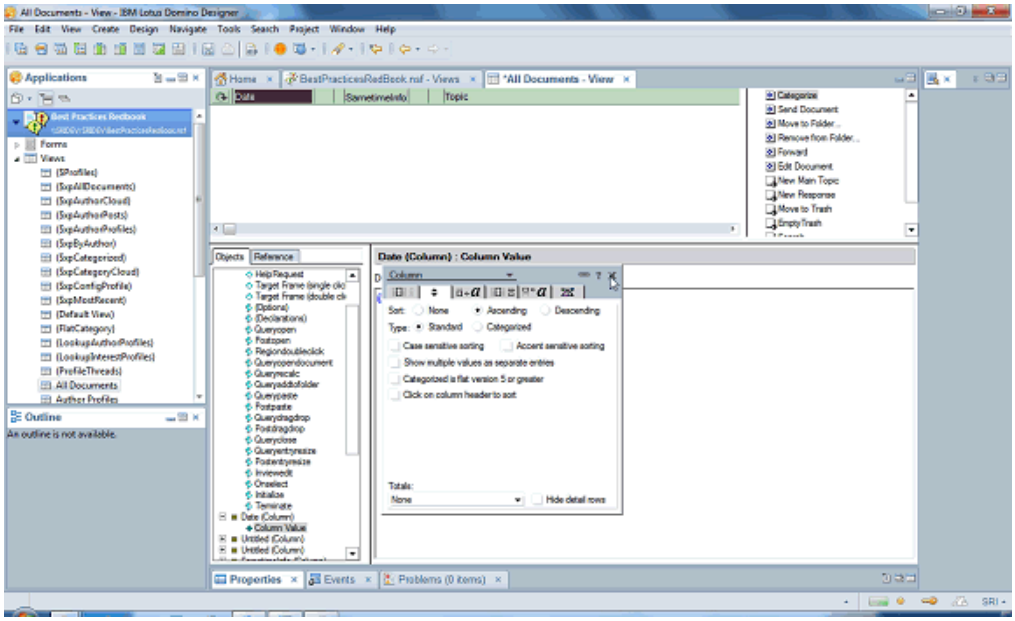
The unsorted view index is 2,612,056 bytes large. This reflects the total size of all the

data displayed in the view columns, plus some overhead for maintaining the list of notes in the view. As we implement features, we see the index change in size, thus, impact the overall performance of our application. The index size isn't an absolute indicator of performance, but as a general rule, the larger it is, the more work it is for the server to keep up to date.

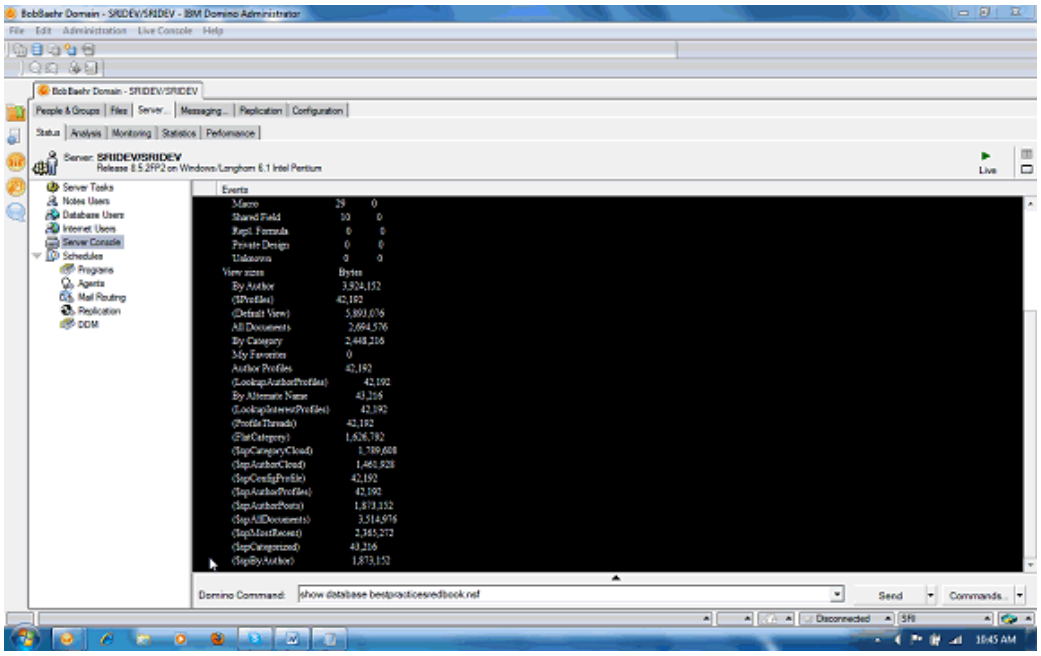
If you have multiple replicas of a Domino application on multiple severs (or clients), each replica generates its own copy of the view index; the index does not replicate. So it's quite possible for the size to vary widely between servers, depending how recently the index has been updated or whether it's ever been generated at all on that server (if not, you would see a size of zero).

Single-column sorting

To show the impact of adding a sorted column to our application, we select to sort the Date column in ascending order:



After saving these changes, closing, and re-opening the database, we see the following size of the view index:

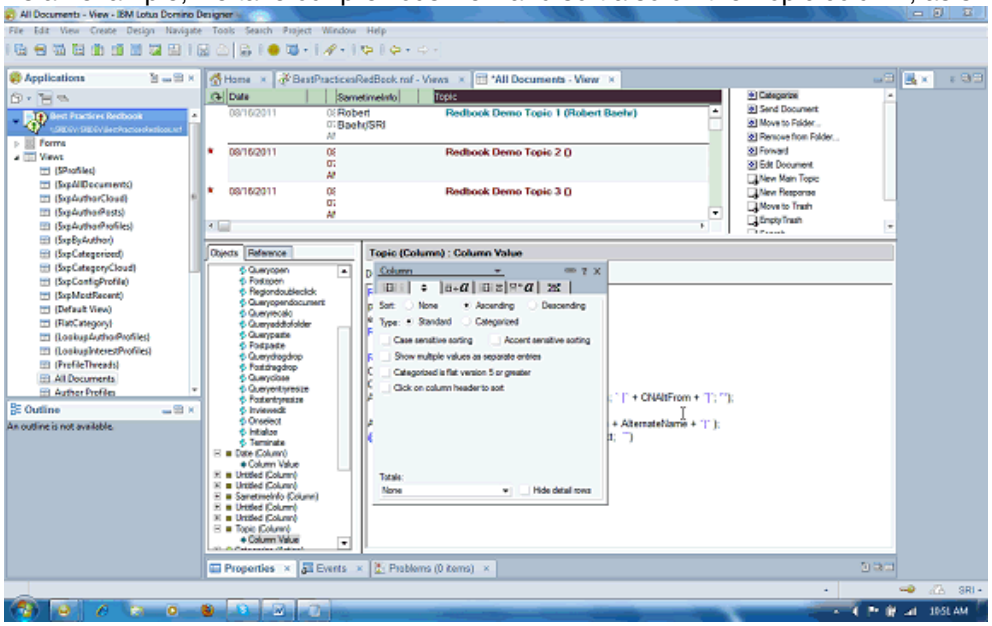


By simply sorting a column, the size of the index increased (in this case, to approximately 2,694,576 bytes). This represents an increase of 3.16% of the size of the view index, compared to the original, unsorted view index size. When we sort a column in a view, we are imposing a slight overhead on Domino with respect to view indexing and performance.

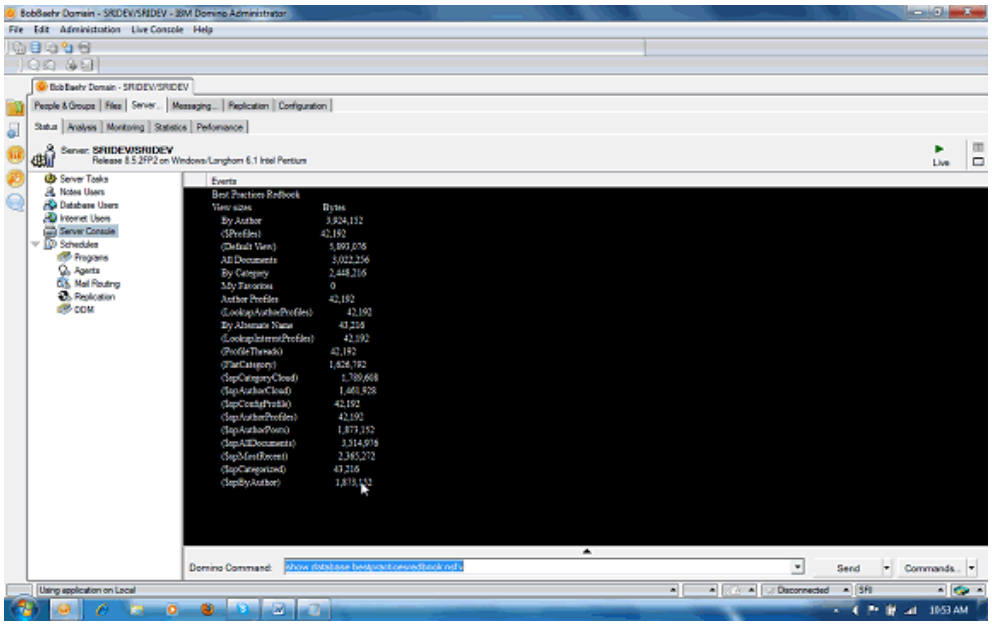
Adding a sorted column to a view does not introduce a large index size increase (all other things being equal), therefore, performance should not be negatively impacted.

Multi-column sorting

Sorting on multiple columns in the same view does increase the size of the view index, because there is more for Domino to consider when parsing and displaying the view. As an example, we take our previous view and sort also on the Topic column, as shown:



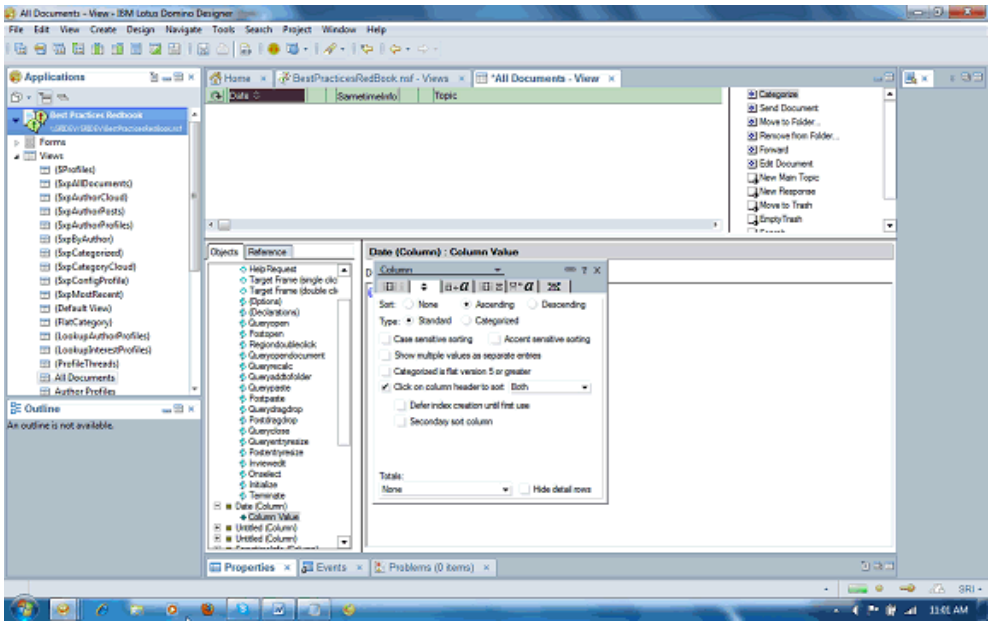
Reviewing the SHOW DATABASE command, we see a noticeable increase in the view index size:



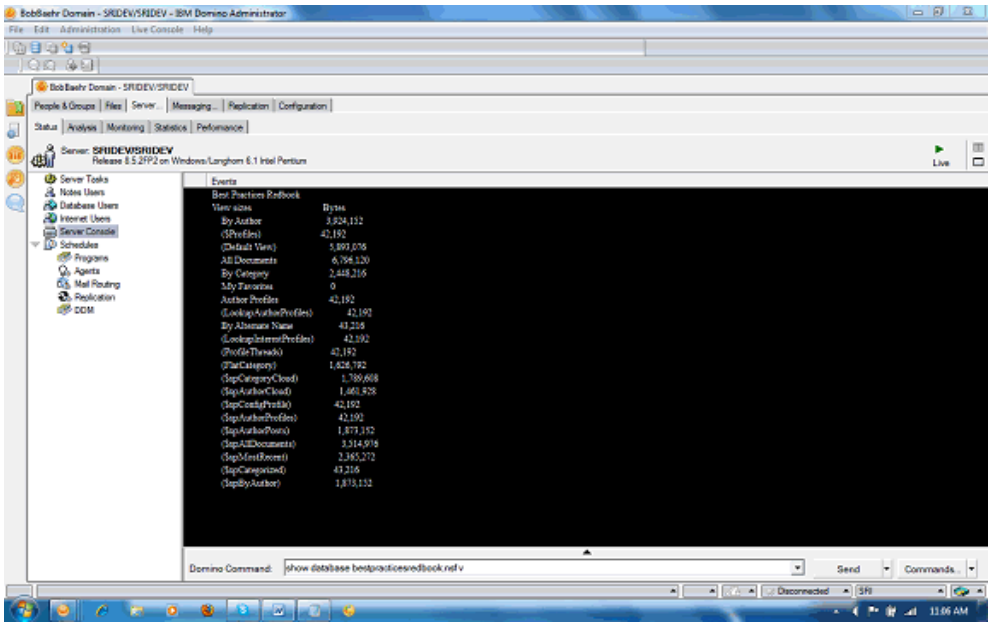
Note that the index did increase in size (up to 3,022,256 bytes), which is a 15.7% increase in the size of the view index. In general, sorting on multiple columns has an impact with respect to view indexing. The performance factor is directly related to the number of documents which display in the view, and the number of columns that are sorted.

Dynamic (user-controlled) sorting

Another option available to Domino developers is to dynamically sort columns, based on the desires of the user. Continuing with our application, we configure the application to allow the user to sort from "hi-to-low" or "low-to-high" for the Date column, as shown:



Upon saving these changes, and, re-opening the application, we see a substantial increase in the view index size (and increase to 6,796,120 bytes), which produces a 160.18% increase in the size of the index compared to the original, unsorted view.



Therefore, our minor design changes have produced the following with respect to view indexing:

- No sort options: Index size is 2,612,056 bytes.
- Sort on one column: View index size is 2,694,576 bytes (increase of 3.16% over non-sorted views).
- Sort on two columns: View index size is 3,022,256 bytes (increase of 15.7% over non-sorted views).
- Dynamic sorting: View index size is 6,796,120 bytes (increase of 160.18% over non-sorted views).

Best practices

It is not reasonable to design a view without utilizing some type of sorting option. Without sorting, the applications would be cumbersome for the end-user, data would not appear in any logical fashion, and the application would, for all intents and purposes, be fairly useless.

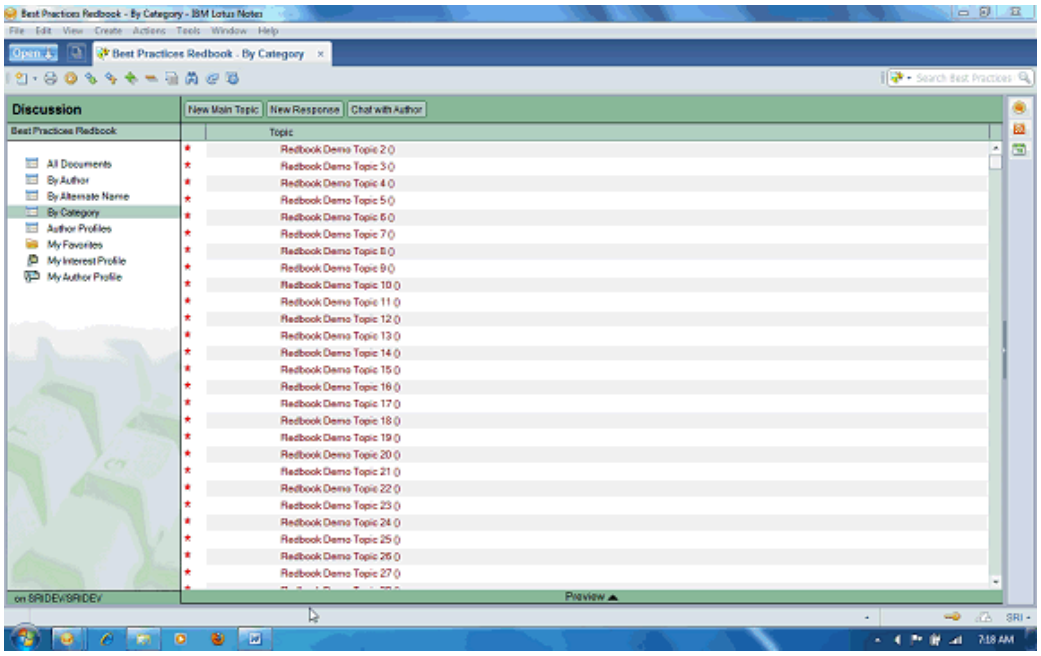
To accommodate performance and usability, consider the following when designing views for your application:

- Consider the view design for performance and scalability.
- Since there is significant performance impact, don't automatically make every column sortable. Use dynamic sorting as needed to give users ways to find documents. Also consider that the performance impact of one resortable view is less than that of two separate views. So, if you can eliminate a view by adding a re-sort to another view, that's a performance win. It's also a UI win, since users are accustomed to being able to click column headings to sort in various applications. It makes more sense to them than having to use an outline to choose among lists of the same documents sorted in a different way. In cases where you do need separate views (for instance, because you want to use categorization) consider using the view column option that lets you link to another view by clicking the column heading. This is intuitive for users, provided the views you link in this way select the same set of documents.
- Learn how to use the view column option "Defer index creation until first use." This minimizes the performance impact of resortable columns by waiting to do the work of sorting them until someone actually uses that re-sort. The first user to request that sort may have a long wait, but once the index exists it will be updated automatically by the server, like regular indexes, so subsequent use is quick. If the index is unused for a length of time, the index will be erased, saving the server work. It's a good choice for columns you want to allow to be dynamically sorted, but which you don't expect to be sorted very often.

6.1.3 Categorization considerations

Categorizing a column (or columns) in a view has been a favorite of Domino application designers for years, allowing us to build views that provide more meaningful data presentation, ease of use, and ease of navigation.

As an example, we look at our Discussion database entries in a simple, flat-list, non-categorized format:

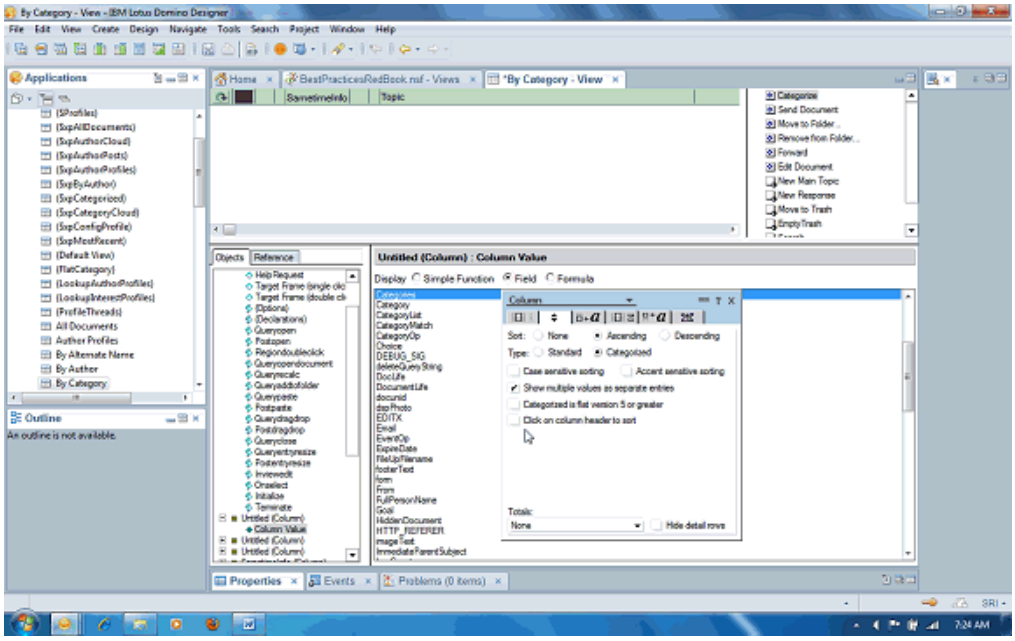

Best Practices Redbook - By Category - IBM Lotus Notes
Best Practices Redbook - By Category
Discussion
New Main Topic
New Response
Chat with Author
Topic
All Documents
By Author
By Alternate Name
By Category
Author Profiles
My Favorites
My Interest Profile
My Author Profile
Redbook Demo Topic 2 ()
Redbook Demo Topic 3 ()
Redbook Demo Topic 4 ()
Redbook Demo Topic 5 ()
Redbook Demo Topic 6 ()
Redbook Demo Topic 7 ()
Redbook Demo Topic 8 ()
Redbook Demo Topic 9 ()
Redbook Demo Topic 10 ()
Redbook Demo Topic 11 ()
Redbook Demo Topic 12 ()
Redbook Demo Topic 13 ()
Redbook Demo Topic 14 ()
Redbook Demo Topic 15 ()
Redbook Demo Topic 16 ()
Redbook Demo Topic 17 ()
Redbook Demo Topic 18 ()
Redbook Demo Topic 19 ()
Redbook Demo Topic 20 ()
Redbook Demo Topic 21 ()
Redbook Demo Topic 22 ()
Redbook Demo Topic 23 ()
Redbook Demo Topic 24 ()
Redbook Demo Topic 25 ()
Redbook Demo Topic 26 ()
Redbook Demo Topic 27 ()
Preview

Although the view we are looking at is called "By Category", we intentionally removed the categorization from this view for the purposes of illustration.

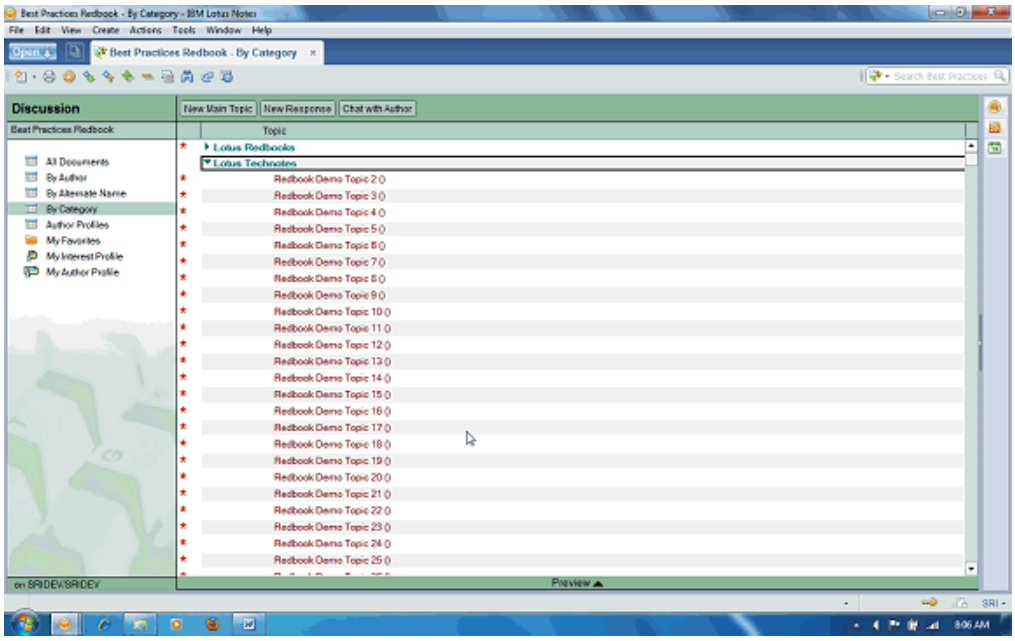
Though a flat-list view presents the documents that we need, it's a little intimidating to the user to have such a long list to choose from. There are a few different ways the user can navigate the view -- by scrolling, by using the search bar to filter the documents, or with the "quick search" where they can just start typing to search by whatever column is currently sorted.

Upon opening the view, Domino must use the view index to gather all information about all the view rows before displaying them on the screen. Compared to a categorized view, a flat view may be slower to open, especially if Readers fields are used in the application.

With the SHOW DATABASE command, the view index size of the above view is approximately 2,448,416 bytes. To see the impact of categorizing columns, we first add a categorized column by using Domino Designer, as shown:

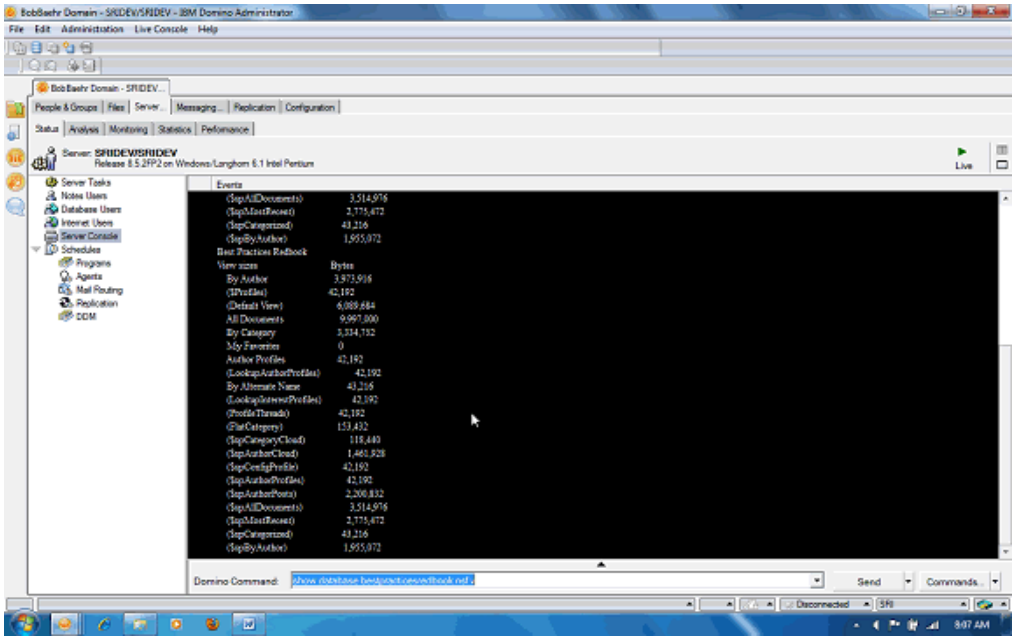


Upon saving the view design, and re-opening the database to the "By Category" view, Domino presents the following view:



The categories shown next to the green “twisty” (Lotus Redbooks, Lotus Technotes) were created along with the document in our sample database; they are part of the data entered into the document when it was edited.

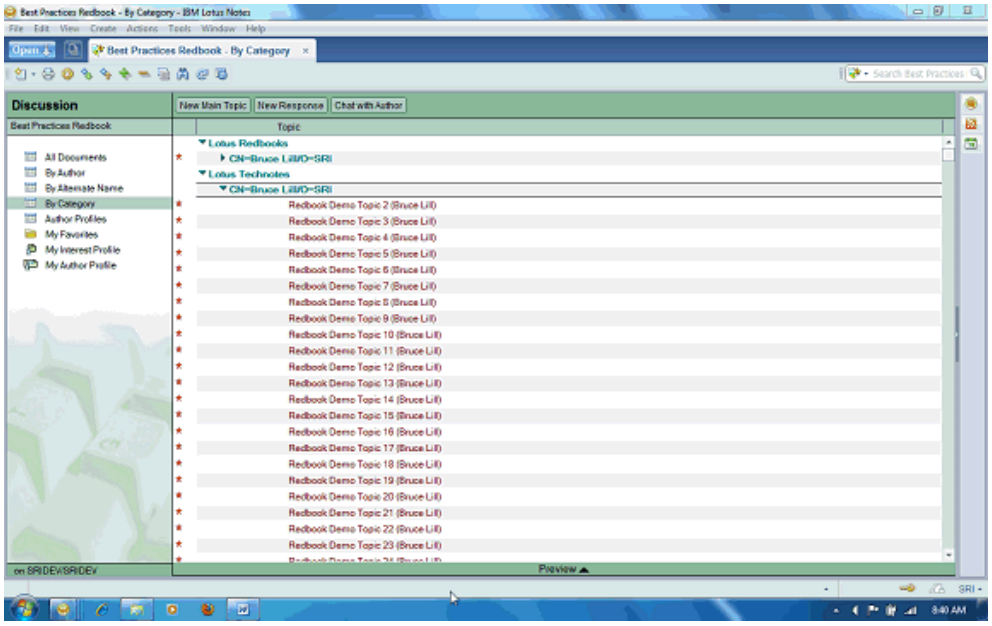
Up until this point, we simply categorized a single view on a single column. The resulting size of the index is as shown:



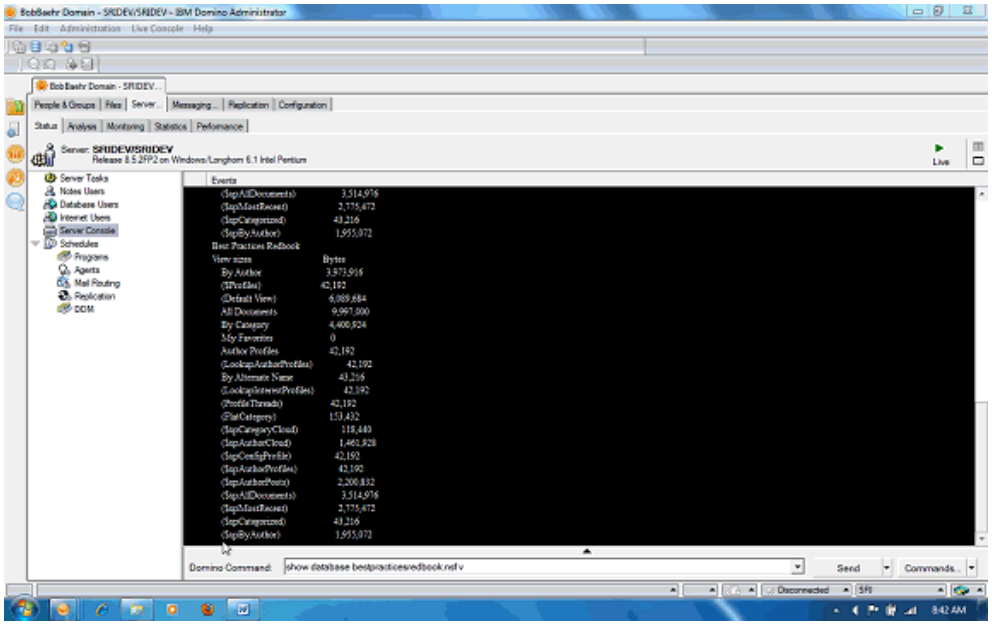
From an index perspective, our view index size increased from 2,448,416 bytes to 3,334,752 bytes – an overall index size increase of 886,336 bytes (or, 36.2%). While this increase may seem large compared to the simple change we made to our view design, we must consider what occurs when we categorize a column:

- A category row is an "entry" in the view table just like a document is. The categorized column in the document row also contains a copy of the category value (it may be different because a category contains one value, whereas the field you're categorizing may contain multiple values).
- If a document does contain multiple category values, there are multiple row entries for that document, which also takes additional space and makes maintaining the list of entries more work.
- Categorization automatically implies sorting, either in ascending or descending order, so there is view index overhead because of the sort option selected.
- When you categorize a column, all documents related to that category are referenced (or linked) back to the category. This information is stored in the view index to provide the “parent/child” hierarchy between the category and associated documents.

If we add a second categorized column to the same view (say, on the Author field), the results are as follows:

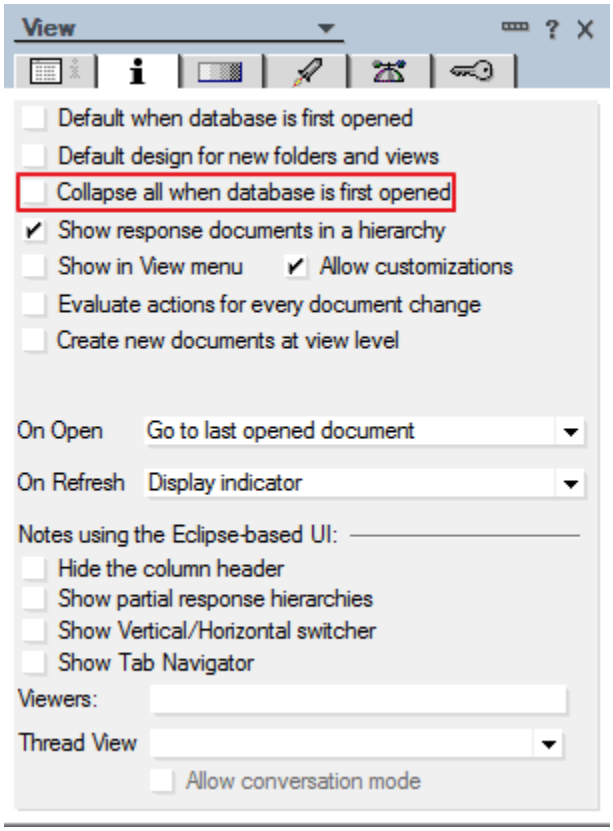

Best Practices Redbook - By Category - IBM Lotus Notes
Discussion
New Main Topic
New Response
Chat with Author
Topic
Lotus Redbooks
CN=Bruce Lill/O=SRI
Lotus Technotes
CN=Bruce Lill/O=SRI
Redbook Demo Topic 2 (Bruce Lill)
Redbook Demo Topic 3 (Bruce Lill)
Redbook Demo Topic 4 (Bruce Lill)
Redbook Demo Topic 5 (Bruce Lill)
Redbook Demo Topic 6 (Bruce Lill)
Redbook Demo Topic 7 (Bruce Lill)
Redbook Demo Topic 8 (Bruce Lill)
Redbook Demo Topic 9 (Bruce Lill)
Redbook Demo Topic 10 (Bruce Lill)
Redbook Demo Topic 11 (Bruce Lill)
Redbook Demo Topic 12 (Bruce Lill)
Redbook Demo Topic 13 (Bruce Lill)
Redbook Demo Topic 14 (Bruce Lill)
Redbook Demo Topic 15 (Bruce Lill)
Redbook Demo Topic 16 (Bruce Lill)
Redbook Demo Topic 17 (Bruce Lill)
Redbook Demo Topic 18 (Bruce Lill)
Redbook Demo Topic 19 (Bruce Lill)
Redbook Demo Topic 20 (Bruce Lill)
Redbook Demo Topic 21 (Bruce Lill)
Redbook Demo Topic 22 (Bruce Lill)
Redbook Demo Topic 23 (Bruce Lill)
Preview

Note that we added the Author (Bruce Lill) across all documents, so that we have the same second-level category value across our application. The results of the view index size are shown below:



Our view index for the By Category view is now 4,400,924 bytes in size, an increase of 31.97% over a single category, and an increase of 43.55% over a non-categorized view. As mentioned earlier, the larger the view index, the more work that Domino has to do to maintain that index, and open the view.

In addition to minimizing the view index size, the **Collapse all when database is first opened** setting on a view can aid performance. This setting is under the view properties when you open a categorized view in Domino Designer, shown below:



The “Collapse all when database is first opened” setting instructs Domino to collapse the "Twisties" when a view is first opened by a user, thus, instructing Domino to display less information to the user upon view open (for our demo application, Domino would show two categories: Lotus Redbooks and Lotus Technotes, as opposed to these categories, the sub-categories, and document column data).

The view will be displayed faster because Domino does not have to initially display as much data to the user. The consideration is that the user must expand a category in order to see the documents it contains.

Best practices

When a design decision calls for the use of categorized views, consider the following:

- Plan categorization carefully, for over-use of categorized views can lead to performance degradation and database size increases.
- If you use categorized views, keep the number of columns per view that you will categorize on as few as possible.
- Use the “Collapse all when database is first use” setting for categorized views.
- Because Domino must maintain each view index, excessive numbers of categorized views may drag down the performance of the server generally.

6.1.4 @Functions and view performance

A view column can contain many different types of data: numbers, text, dates, times, and formulas. When you open a view, Domino must apply the following to display the view:

- The view selection formula that Domino uses to determine which documents to display in the view.
- Column formulas that are used by Domino to determine the data to display in the columns for each document.

The Domino server tries to save work by storing the view index, including the values of all columns for each document indexed. When the contents of the view are needed -- for instance, when a user tries to open the view -- the server only has to look at any documents which were created or modified since the view index was last updated, to decide whether they should be in the view and update or insert the row entries for those documents.

But this cannot work if the selection or column formulas are such that they might give a different answer at different times even if the document has not been modified. This can occur if the selection formula or a column formula uses any of the following functions:

- @Now
- @Today
- @Yesterday
- @Tomorrow
- @Accessed

If any of these functions are used, the server cannot assume that the value it calculated previously is still valid. So, every time the view is used, it creates the view index from scratch, evaluating the selection formula against every document in the database, then calculating the column values for each matching document. Naturally, this takes much longer than just examining recently modified documents.

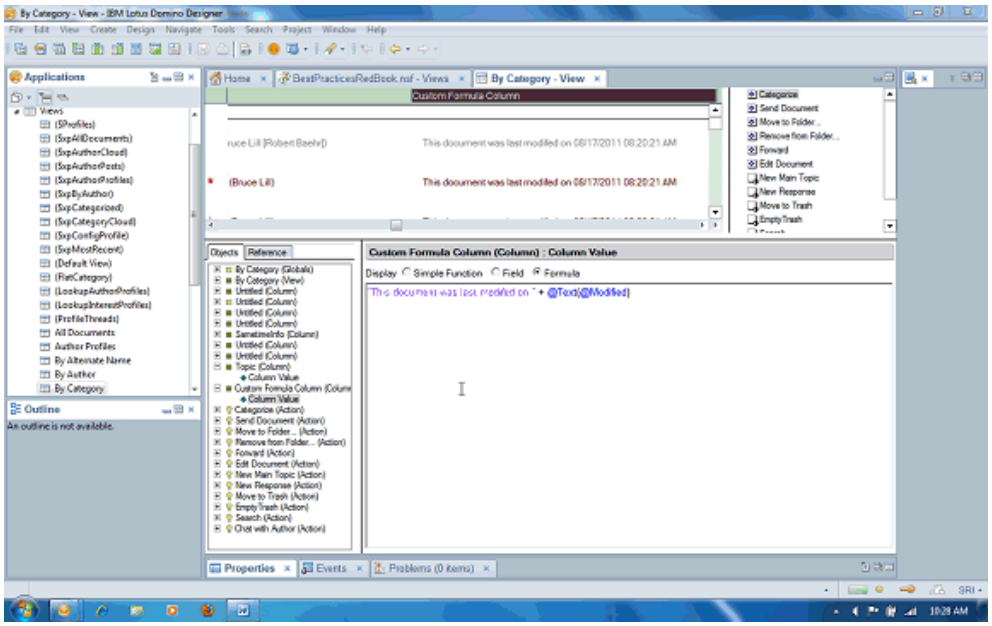
Some notes regarding this situation:

- This rule applies only to column and selection formulas. Other formulas associated with the view, such as column hide formulas, action button formulas, and so on, are not used in calculating the view index, but are instead calculated by the client when the view is used. You can use @Now in those without any problem.
- It would seem to make sense for the Domino server to distinguish between the use of @Now, which *always* returns a different value, versus @Today, which returns the same value for 24 hours at a stretch. If only @Today is used, the server really only needs to recalculate the index from scratch once per day. However, that's not how it works; @Today is just as bad for performance as @Now.
- You may have come across a tip that tells you to use @TextToTime("today") instead of @Today, for better performance. This is incorrect. You do get better performance, but the view contents are incorrect. All this does, is fool the view indexer into not realizing that you're referencing the current date in your formulas. So, the indexer doesn't know that it has to re-evaluate the formulas against documents that weren't modified, and the values in the view get out of date.
- The @Random function returns a different value every time it's called, but the view indexer doesn't include it in the functions that force an index rebuild on each use. That's probably because using the @Random function in a view is kind of weird

anyway, and it's not clear whether there's really a need to re-evaluate it for old documents.

- There are some functions that you might think would be changeable over time, but really are not. As an example, the @DocNumber function; if there are three documents in the view, the doc number for the last document is 3; if you create a new document, and the view sorting puts it at position 2, the old document number 3 becomes document number 4, even though that document hasn't been modified. There's a trick with @DocNumber and similar functions, though; they actually return a static string with special codes in it that instruct the client to calculate a value and insert it at that point. So the value in the server index doesn't change because of changes to other documents.
- There are several other functions, listed in the documentation, that don't work in views at all. They will not force a rebuild, but they also don't do anything useful.
- Any other functions that return or manipulate date/time values, are not a problem. Feel free to use @Modified, for instance.

As an example of that last point, consider the following formula added to the By Category view:



The formula used in the "Custom Formula Column" is as follows:

```
“This document was last modified on ” + @Text(@Modified)
```

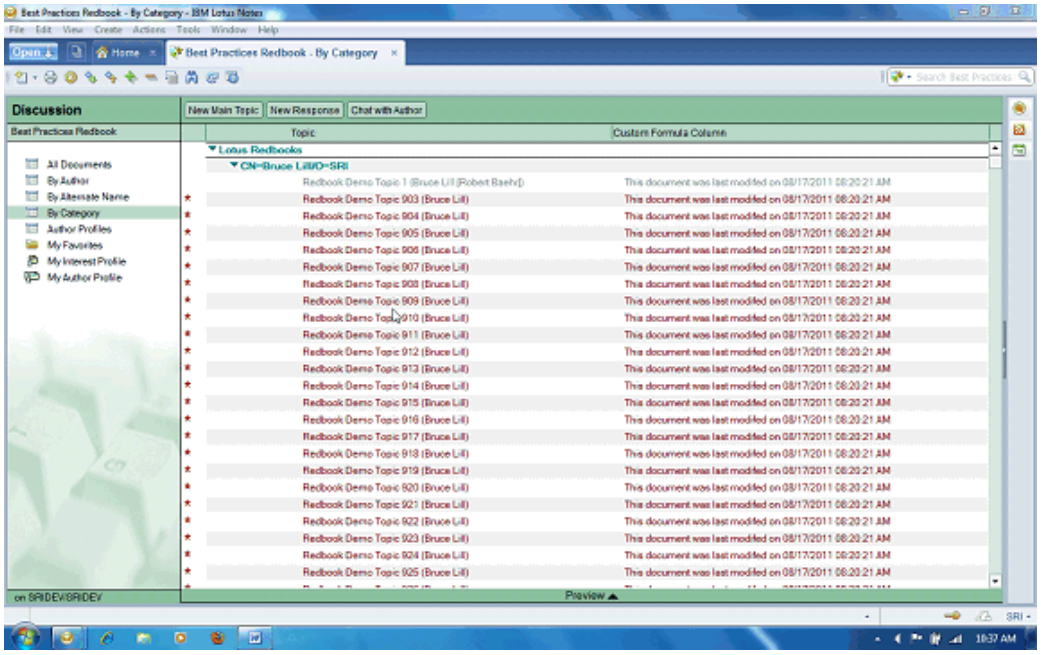
The formula concatenates the string "This document was last modified on" with the result of the @Modified function, converted to text.

NOTE: We discourage using @Text to convert numbers and dates in view columns, because this uses the server's formatting settings, overriding the user's formatting preference. This formula is just an example.

When writing column formulas, no matter how complex, you need only look for the five

functions listed earlier, to determine whether they will force an index rebuild on every use. In this case we're safe; neither @Text nor @Modified cause a problem.

In the resulting view, we see the following:

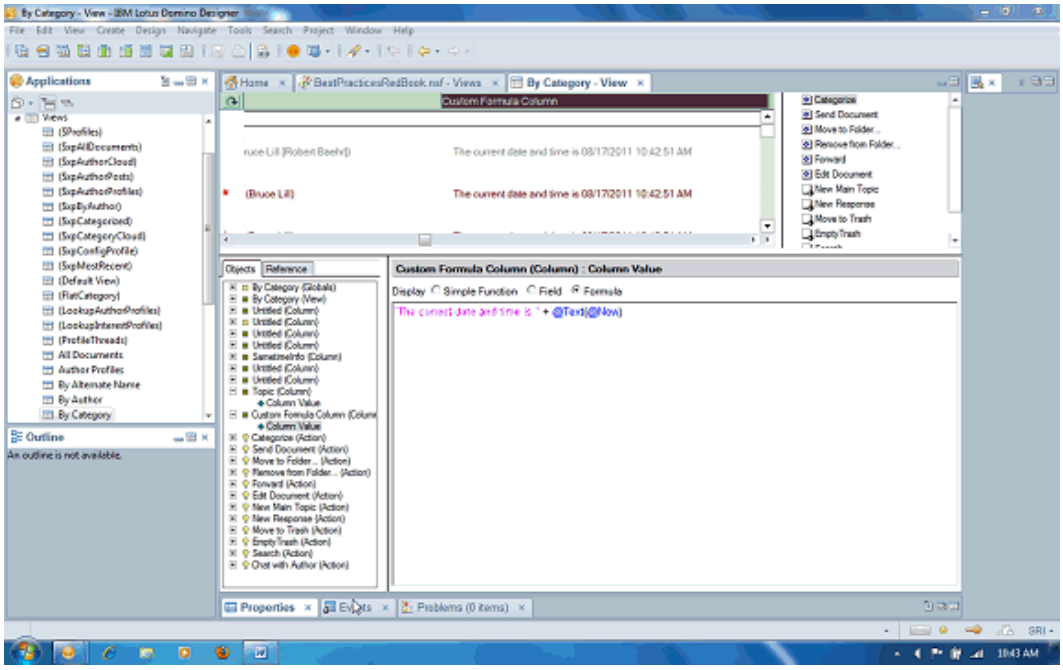

Best Practices Redbook - By Category - IBM Lotus Notes
Discussion
New Main Topic
New Response
Chat with Author
Topic
Custom Formula Column
Lotus Redbooks
CN=Bruce Litt/O=SRI
Redbook Demo Topic 903 (Bruce Litt)
Redbook Demo Topic 904 (Bruce Litt)
Redbook Demo Topic 905 (Bruce Litt)
Redbook Demo Topic 906 (Bruce Litt)
Redbook Demo Topic 907 (Bruce Litt)
Redbook Demo Topic 908 (Bruce Litt)
Redbook Demo Topic 909 (Bruce Litt)
Redbook Demo Topic 910 (Bruce Litt)
Redbook Demo Topic 911 (Bruce Litt)
Redbook Demo Topic 912 (Bruce Litt)
Redbook Demo Topic 913 (Bruce Litt)
Redbook Demo Topic 914 (Bruce Litt)
Redbook Demo Topic 915 (Bruce Litt)
Redbook Demo Topic 916 (Bruce Litt)
Redbook Demo Topic 917 (Bruce Litt)
Redbook Demo Topic 918 (Bruce Litt)
Redbook Demo Topic 919 (Bruce Litt)
Redbook Demo Topic 920 (Bruce Litt)
Redbook Demo Topic 921 (Bruce Litt)
Redbook Demo Topic 922 (Bruce Litt)
Redbook Demo Topic 923 (Bruce Litt)
Redbook Demo Topic 924 (Bruce Litt)
Redbook Demo Topic 925 (Bruce Litt)
This document was last modified on 08/17/2011 08:20:21 AM
Preview

Once the view is displayed, you do not have to refresh view index until the following situations occur:

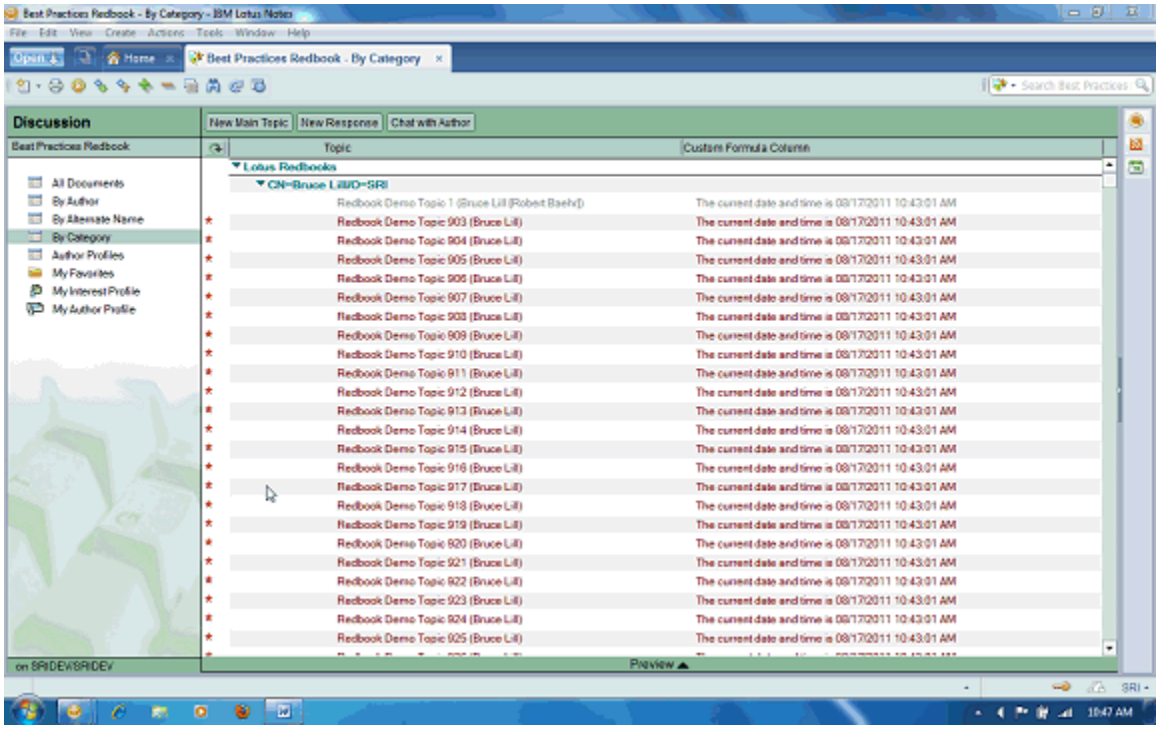
- One of the underlying documents is modified.
- Domino performs maintenance on the view index.

Suppose that we make the following modification to our column formula:

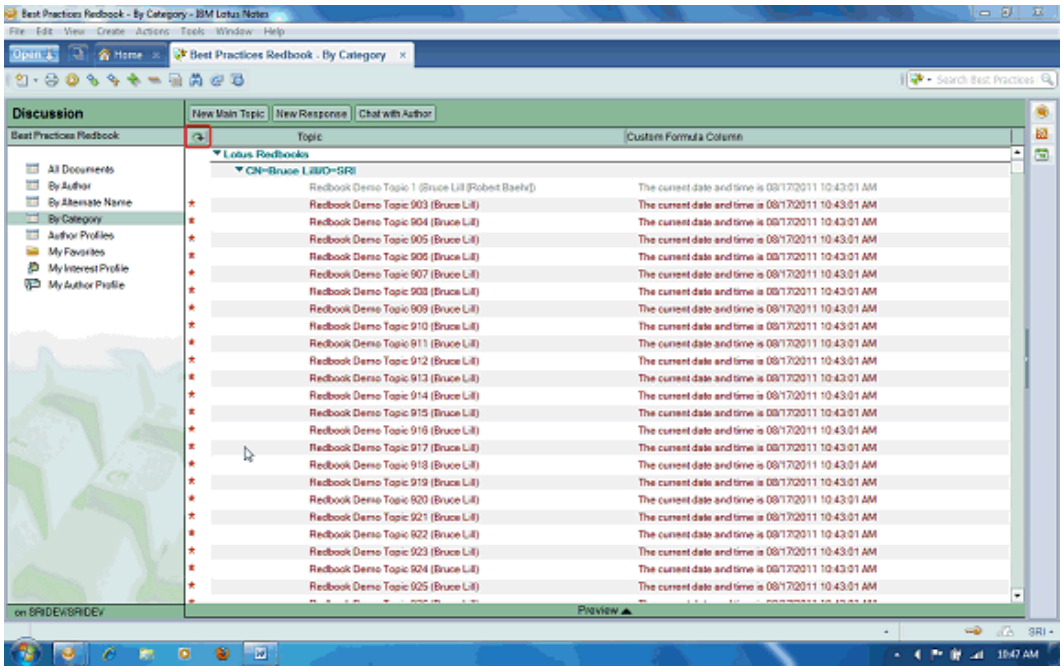
```
“The current date and time is ” + @Text(@Now)
```



Now that we're using the dangerous @Now function, the view will be slower. There's also a visual indication that the view index is considered always out of date, as shown below:



Observe the "view refresh" icon located in the upper-left corner of the view (shown in the red box below).



This is an indicator that something in the view has changed, requiring an update to the view index. With normal views, this only occurs if the client notices that some documents have been modified since the view was opened. In this case, though, what has changed is @Now. In fact, it is always changing, returning the current date and time (including seconds and hundredths). This view is in a constant "needs refresh" state.

Best practice

When using @Functions in a view column formula or view selection formula, avoid the functions @Now, @Today, @Yesterday, @Tomorrow and @Accessed.

So then, what do you do if you want a view of "documents created today?" We want to avoid the obvious way of comparing @Created and @Today in the selection formula. There are a few ways to manage this, depending on how users need to use this view and how much work you're willing to do.

- Create a view that simply sorts documents by the date/time value, so that the documents of interest appear at the top of the view. Of course, the view also will contain a lot of documents you didn't want there, but it may be an acceptable compromise for best performance.
- Create a view categorized by date, and use the "single category" function to display only today's category.
- Use a folder instead of a view. Write an agent that executes nightly on one server, shortly after midnight, and adds and removes documents in the folder. Folder membership replicates, so it need not run on all servers. Depending what the selection criteria are, you might also have to have a "new and modified documents" agent to assign folder membership for documents created during the day.
- If the date-related formula is in a column rather than in the selection formula, use a "user programmable" column to read the formula from a profile document. Write a nightly agent to update the formula to contain today's date as a hard-coded value.

Use @Date function to express a date in a way that doesn't depend on the date formatting settings of the server or client (for a local replica).

- The user-programmable column option can also be used for selection, though it is a bit of a kludge. If a column formula uses @Return, the document is excluded from the view.

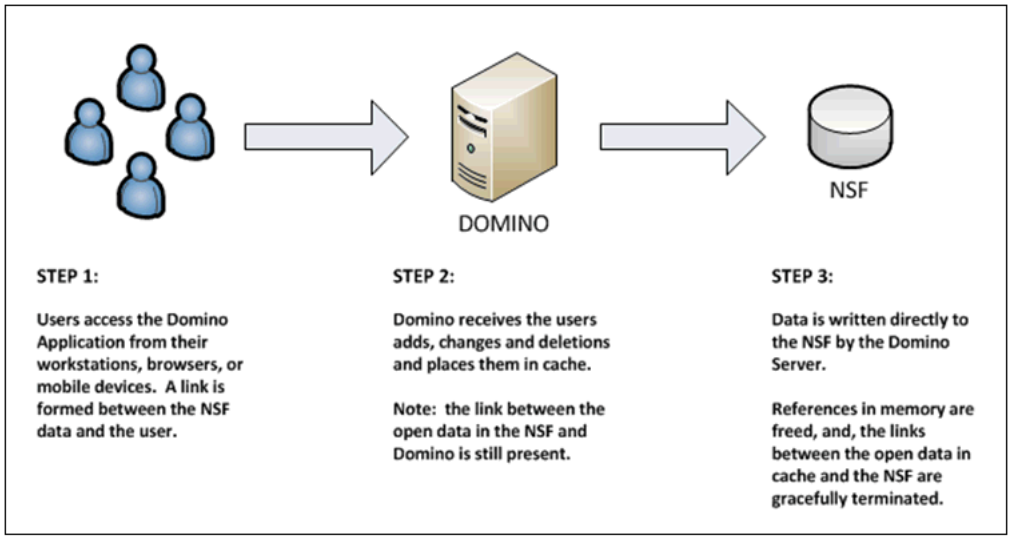
6.1.5 Views and transaction logging

Important note: The use of transaction logging as part of any Domino application design requires consultation with the Domino Server administration team prior to implementation.

The best way that we can describe Transaction Logging is through illustration.

Without transaction logging in Domino

Without the support of the transaction logging in Domino, a server-based Domino application is manipulated in the following way:

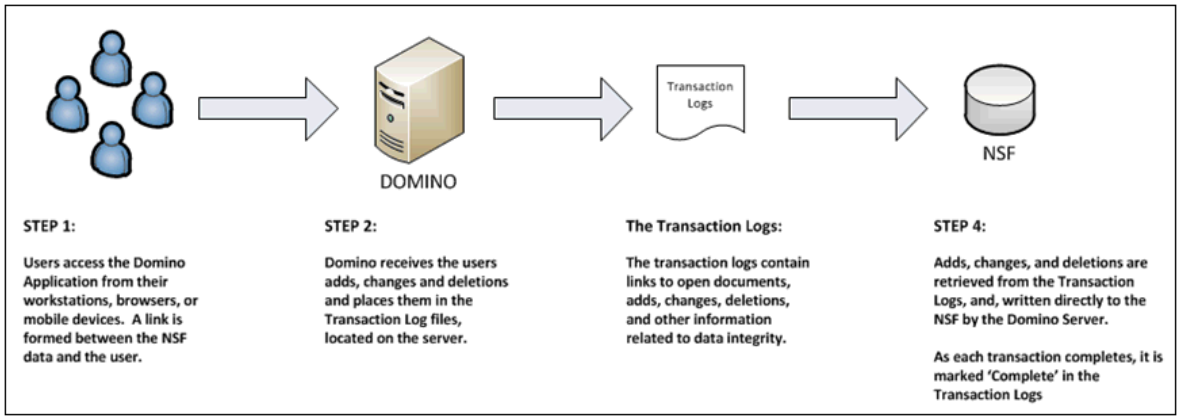

DOMINO
NSF
STEP 1:
Users access the Domino Application from their workstations, browsers, or mobile devices. A link is formed between the NSF data and the user.
STEP 2:
Domino receives the users adds, changes and deletions and places them in cache.
Note: the link between the open data in the NSF and Domino is still present.
STEP 3:
Data is written directly to the NSF by the Domino Server.
References in memory are freed, and, the links between the open data in cache and the NSF are gracefully terminated.

This introduced many potential problems, especially in the event of a server crash. These problems include:

- In the event of a server crash, the Domino application is left in an 'inconsistent state'. In short, the data in cache was lost, and, the data in the NSF was not written to completely (between steps 2 and 3, above). Thus, upon restart, the server had to perform consistency checks on each application. The consistency checks means that the application could be off-line for minutes to hours, depending on the size and complexity of the application.
- In addition, since the application state is inconsistent, each view index has to be rebuilt, which, depending on the size and complexity of the application, could take minutes to hours.

With transaction logging

Since introducing transaction logging, Domino has become much more stable and reliable than ever. Consider the processing of data when transaction logging is enabled:

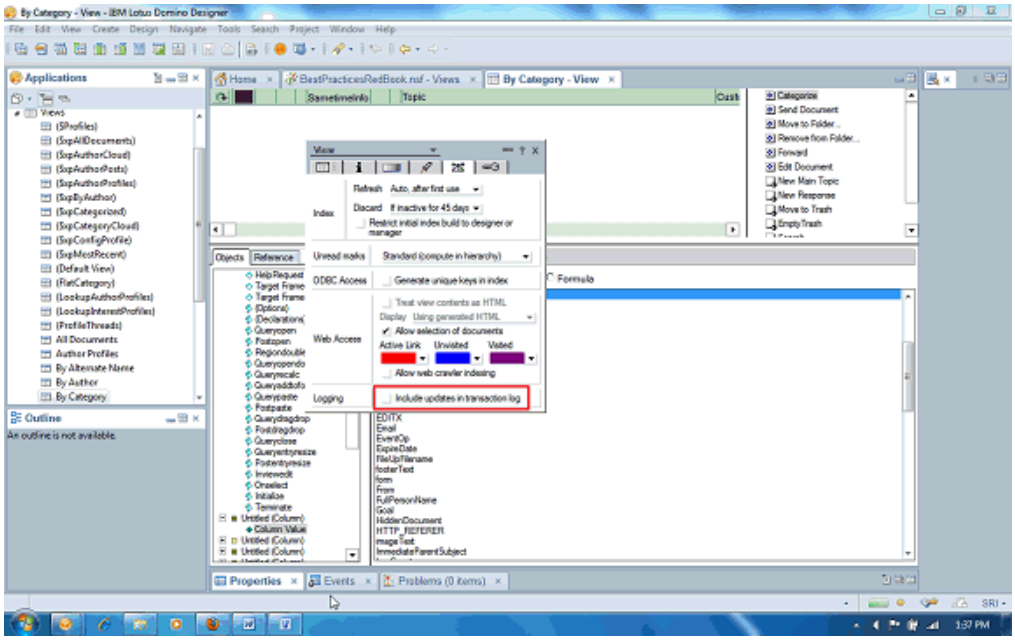

Transaction Logs
NSF
DOMINO
STEP 1:
Users access the Domino Application from their workstations, browsers, or mobile devices. A link is formed between the NSF data and the user.
STEP 2:
Domino receives the users adds, changes and deletions and places them in the Transaction Log files, located on the server.
The Transaction Logs:
The transaction logs contain links to open documents, adds, changes, deletions, and other information related to data integrity.
STEP 4:
Adds, changes, and deletions are retrieved from the Transaction Logs, and, written directly to the NSF by the Domino Server.
As each transaction completes, it is marked 'Complete' in the Transaction Logs

In the event of a server crash, Domino will restart, performing the following:

- Domino opens the transaction logs, looking for any transactions not marked as **Complete**.
- If the entire transaction was written to the log, then Domino writes the data into the NSF, marking the transaction in the log as "Complete". This process is called a "**roll forward**".
- If the entire transaction was not written to the log, then Domino puts the data in the NSF back to the state it was in before that data was edited, and, mark the transaction 'Failed'. This process is called a "**roll back**".

The important piece for us to understand is that Domino will, when transaction logging is enabled by the Domino Administrator, protect the data from corruption, getting the server back on-line as fast as possible. However, there is one more piece we have to consider: *view index*.

View logging does not impact the view indexing performance unless you explicitly design your views to do so. By default, the view logging is disabled on a view that you build using Domino Designer. The view logging is enabled through Domino Designer, view properties, on a per-view basis, as shown:



Once enabled, view indexes can now take advantage of Domino transaction logging, assuring that the database is brought back on-line fast, and, the indexes are updated without a full-scale view index rebuild. Combined with database transaction logging, a production server can be on-line in minutes after a server crash (instead of hours).

Best practices

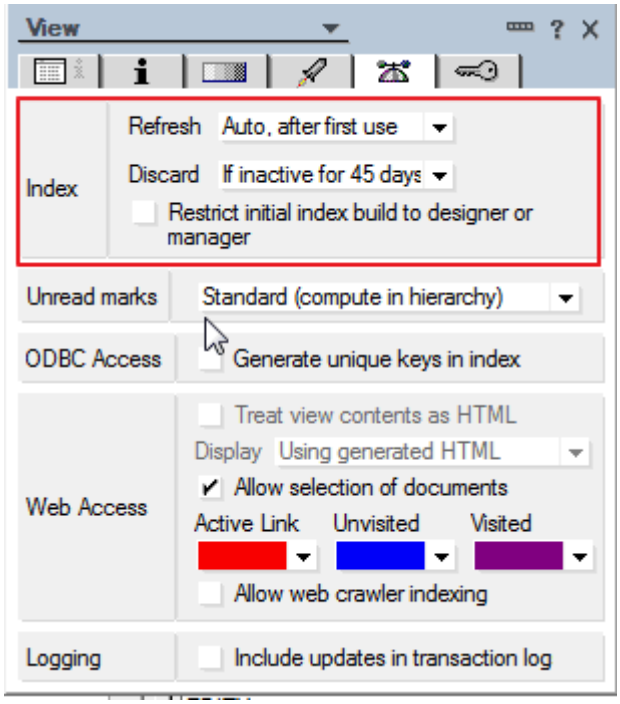
When using transaction logging and view logging in developing your Domino application, consider the following:

- Transaction logging must be enabled and configured by the Domino Administrator before view logging can be successfully enabled.
- View logging does incur additional overhead on the Domino Server, so work with your Domino Administrator to make sure that the server resources (memory, CPU, and disk) can adequately process view logging requests.

6.1.6 Controlling Domino view index maintenance

From the vast amount of information we have learned thus far, we see that view indexes are an important part of Domino application functionality and design. Most of the maintenance of the index is performed by the Domino server (specifically, the UPDATE and UPDALL server tasks). The application designers have limited control over the view indexing process, on a per-view basis. The application developers can decide how the index is refreshed, how it is maintained, thus impacting the performance of the application.

Each view independently has settings that you can explore, using Domino Designer, which have a direct impact on view indexing, thus, performance of your application. In the view properties of the Domino Designer, select the Advanced tab. The options in the Refresh dropdown control the view indexing:



Each index option of the Refresh dropdown is described below.

Auto, after first use
This option, which is the default for new views, waits to create a view index until the view is opened for the first time. After that, the index is kept up to date by the UPDATE process, unless it should become inactive for a period of time (which you can specify in the Discard field). At that point the view index is discarded, and no longer kept up to date, until someone uses it again. The server also updates the view index every time the view is opened, to make sure to catch any changes made since the last UPDATE pass.

- Benefit: Active views are kept reasonably up to date, so that users don't have to wait much for the view to open. Inactive views eventually stop causing extra work for the server, so that unused databases don't slow your server merely by existing on the disk.
- Consideration: Once users start using the view, its existence causes work for the server even if the users aren't using it often. Enough views of this type that are in occasional use, can drag down server performance.

Automatic
The UPDATE process will create an index for this view, and keep it up to date, whether anyone is using the view or not. In addition, anytime the view is opened, the server updates the view index to make sure of catching recent changes.

- Benefit: The view is always up to date, and opens quickly for the user .
- Consideration: The presence of this view causes work for the server even if nobody ever uses the view. Only select this option if performance is a key concern for that specific view.

Manual
The view index is never generated or updated automatically. It is created the first time the view is used, but *not* updated every time a user opens the view. This option allows the

view to be opened very quickly because the stored index information is displayed without checking for recent modifications. Since the view index is not refreshed, the data displayed in the view might not match the data currently in the documents. The user will likely see the view refresh icon ( ) when they open the view, advising them that the data are out of date. They can refresh the view by clicking this icon or pressing the [F9] key.

- Benefit: The view open quickly, a feature especially useful in large databases.
- Consideration: The data may be stale. Refreshing the view index is a manual operation for the user.

This is a good choice for views whose contents change seldom or never, such as help documents.

Auto, at most every
This option allows you to control the maximum hourly interval at which the view index will be refreshed. When selected, the application designer must enter a number of hours between view index refreshes. Once applied, the view index is refreshed automatically, but not more often than the number of hours entered. For example, if you set the Auto, at most every option to four hours, the UPDATE process will refresh the index if it's been four hours or more since the last update. Like Manual updating, the view index will not be updated when a user opens the view, but they will see the refresh icon and can manually refresh. The time between updates may also be more than the selected interval, since UPDATE is a polled process that scans through all databases repeatedly, and if there's a lot of work to be done in other databases, it can fall behind.

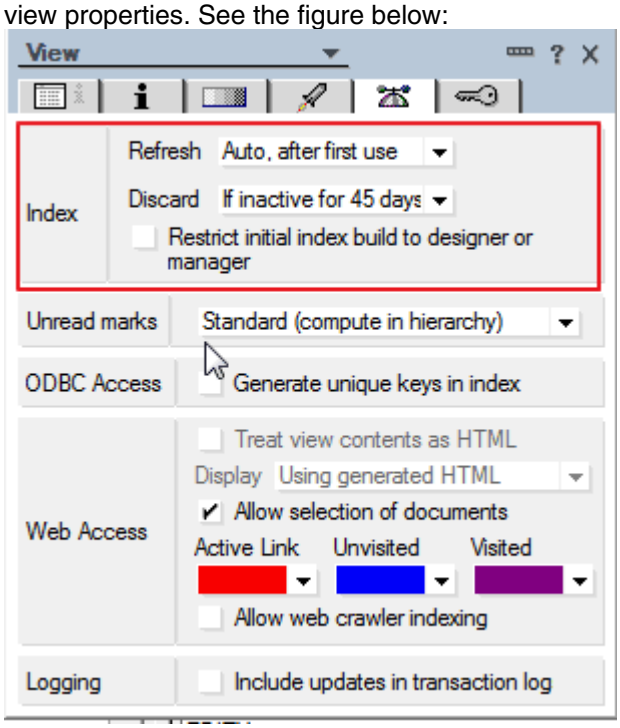
- Benefit: The view index is automatically refreshed at controlled time increments. This can decrease indexing time, and assure that server resources are not being consumed by a more frequent refresh interval. The view also opens very quickly.
- Consideration: When you open a view, the view index is likely stale, and may need to be refreshed manually.

Because of the combination of rapid access and eventual updating, this is a good choice for views whose contents change only occasionally, such as keyword documents accessed by @DbLookup. Even though the database is active generally, we don't need to update this index with every use; it's acceptable if it takes a few hours in exchange for better performance overall.

“Discard” Index Option
By default, Domino performs an incremental "refresh" of the view index, looking for new and changed documents since the last refresh and inserting their information among the existing view entries, or deleting existing entries if the document no longer qualifies for inclusion. You have the option to “rebuild” the view index, destroying the existing index and creating a new one from scratch (you can do this manually from the view via Shift+F9, or via calls to the Notes APIs). Naturally, unless there have been a huge number of modifications, a refresh is much faster than a rebuild. While it may sound attractive to "start fresh," you can generally rely on it that the index for any documents that haven't been modified, is correct and up to date. Use refresh and don't worry about it.

To control how often the view index is rebuilt from scratch, use the Discard options in the

view properties. See the figure below:



If you "discard" the view index, Domino must create the index from scratch the next time someone wants to use the view. This doesn't make the view more reliable, only slower to open. The advantages of discarding the index are that it's not taking up space on disk, and the UPDATE process no longer has to update it, so that frees up server resources.

The options for automatically discarding the view options are described below:

If Inactive For 45 Days
This is the default option. If Domino finds that a view is inactive (no users access) for 45 days, it deletes the view index from the application. This can save storage space on infrequently accessed views. 45 days is a good length of time for most views; if they become disused for that long, there's a good chance that they'll continue to be disused for much longer, so there's no point in expending resources to keep them updated.

Note: this option is ignored if the Refresh setting is "Automatic".

After Each Use
This setting instructs Domino to not store the view index at all, but to calculate it from scratch every time the view is opened. This is rarely a good choice, but might be appropriate in some special cases. This is the same behavior that occurs automatically if the selection or column formulas use @Now; we know that any stored view index is immediately invalid because @Now is not the same the next time we look at the view.

If Inactive For
This option is similar to the "If Inactive For 45 Days" setting, except you get to define the number of days between view index discards. If Domino finds that a view is inactive (no users access) for the number of days specified, it will delete the view index from the application. This can save storage space on infrequently accessed views, or in case you know enough about the usage pattern of the view to select a more specific number than the default 45 days. For example, a view containing sales data that's needed once per quarter to prepare a report; once it's been inactive for a week, we know they're done with

it until next quarter.

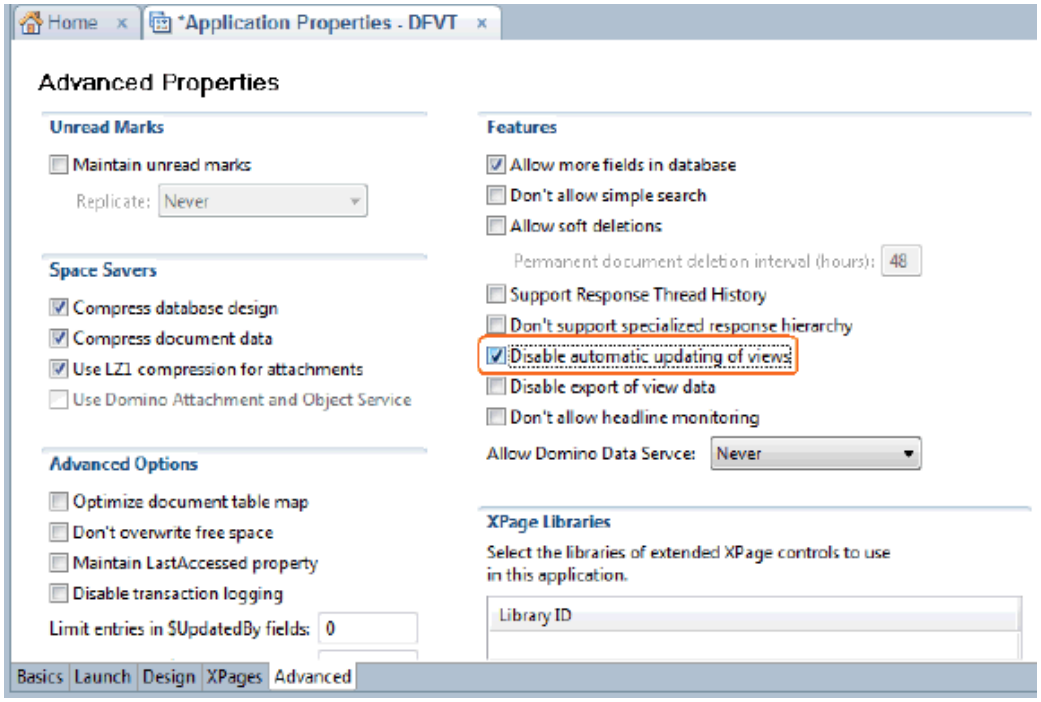
Note: this option is ignored if the Refresh setting is "Automatic".

Best practices for Discard settings

- Given the important relationship between view indexing and performance, first try to minimize the number of views when designing your Domino application.
- For those views you do need, at least consider whether the default indexing options are appropriate.
- In existing applications, if you suspect views are unused but aren't sure enough to just delete them, set their discard settings to a low number of days to reduce the performance penalty if someone opens it by accident.

The Database Option "Disable automatic updating of views"

This option instructs the UPDATE process to skip this database entirely. It's a good choice for archives and other databases that are not often used. Views with automatic update options will still be updated when the user opens them, but they will never be updated or created automatically.



6.1.7 View Externalization

Watch this space.

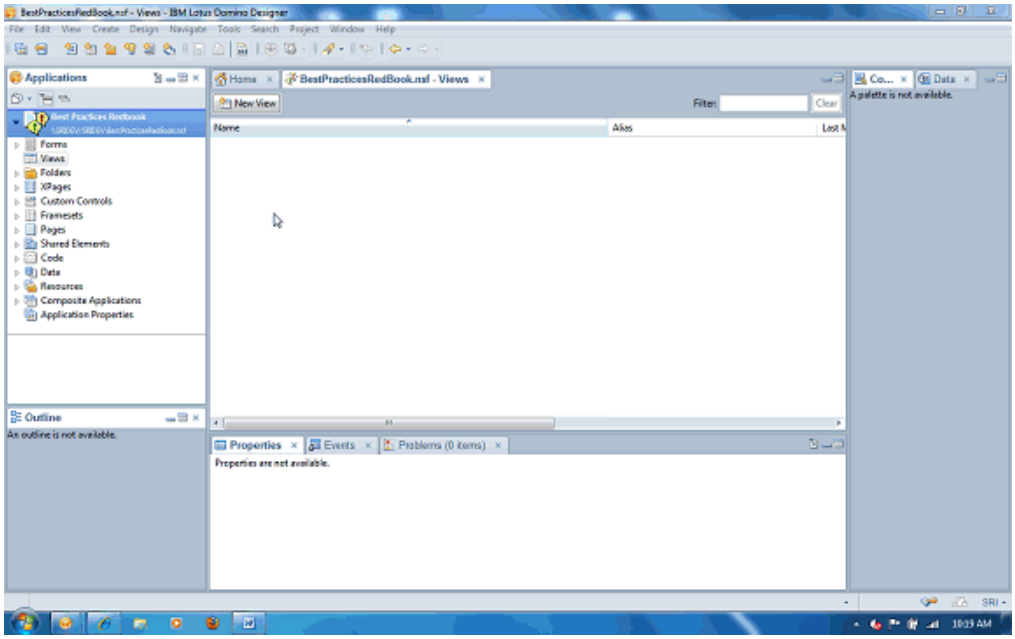
6.1.8 XPages and views: Your application just got a whole lot faster

Introduced with Release 8 of Lotus Domino, XPages offers a new application design paradigm over traditional Domino applications. While the benefits of XPages are many, we focus on how XPages can help you develop “faster applications faster” with respect to view design.

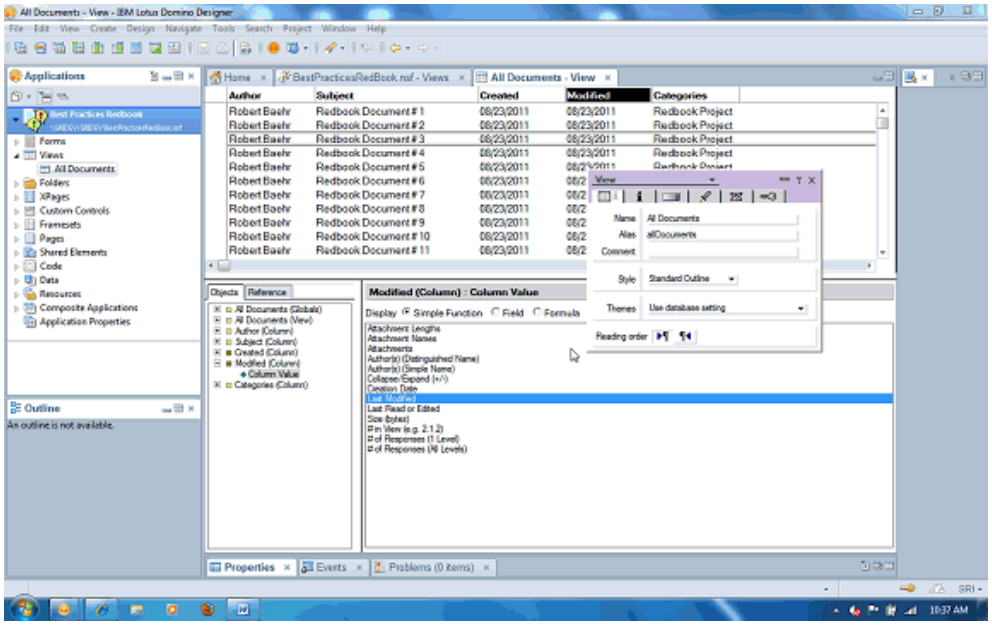
To minimize the impact of view design on our application, wouldn’t it be nice if we could build the same application, with only one view?

In this section, we use the sample application that we have been discussing to demonstrate how to create customized views for an application without creating multiple physical views using Domino Designer. With this process, you minimize design time, view indexing time, disk space utilization, while maximizing performance.

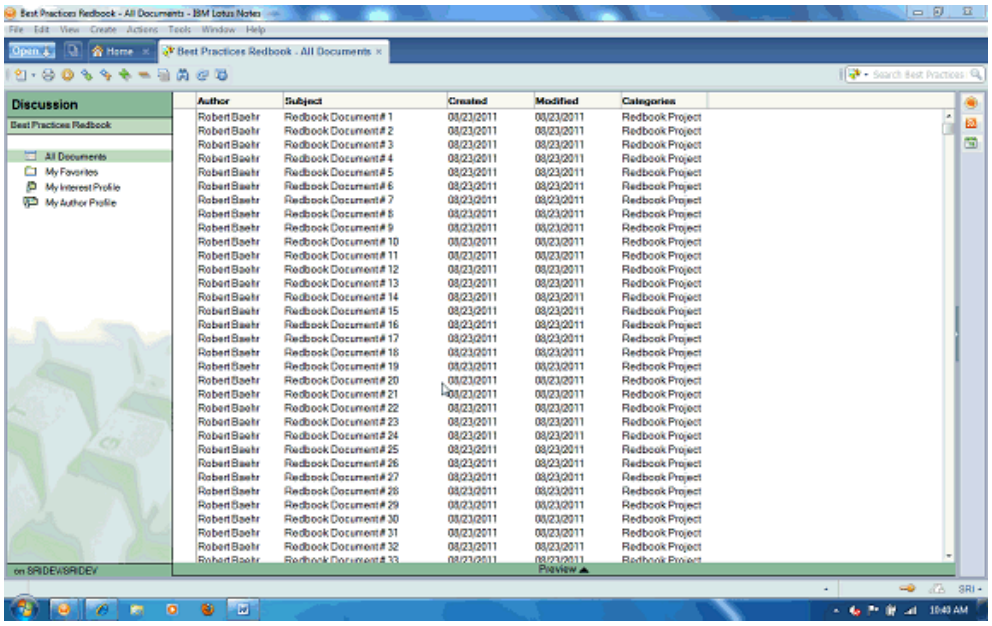
While it is not consistent with real-world applications to contain only a single view, the discussion database has that capability if XPages are used as the interface component. The objective here is to minimize the number of views in the application, thus, to create a faster performing application. In this example, we have intentionally deleted all views from the database and use XPages to provide us with the interface to reconstruct much of the functionality of the original application, using fewer views. See the figure below.



You start by designing a single view, containing all of the rows and columns from our discussion database, enabling all desired sorting options for each column, as shown:



We now have an All Documents view, which contains all documents for our sample application. Looking at the view from the Notes Client, we have all documents displayed in a flat, non-productive format:



We then enable column sorting and categorization on the columns that we want to be displayed in a sorted or categorized format in the application. Though this adds overhead to the view index, because the application has only one view, the view indexing is minimized because we don't need multiple views, and that can increase performance. XPages applications are able to take advantage of the "click to sort" column options to quickly re-sort data themselves.

This single-view application has the disadvantage in that it lacks:

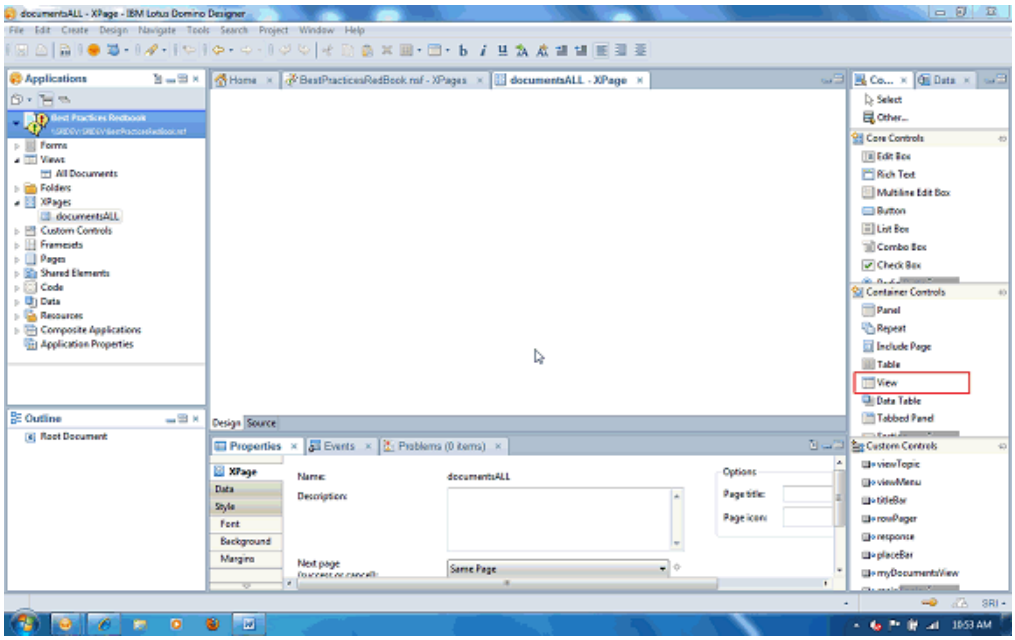
- Logical information ordering
- Ease of navigation
- Robustness that users expect in modern applications

We now explore how XPages can give us multiple, useful views of the data presented above without physically designing another view in the application.

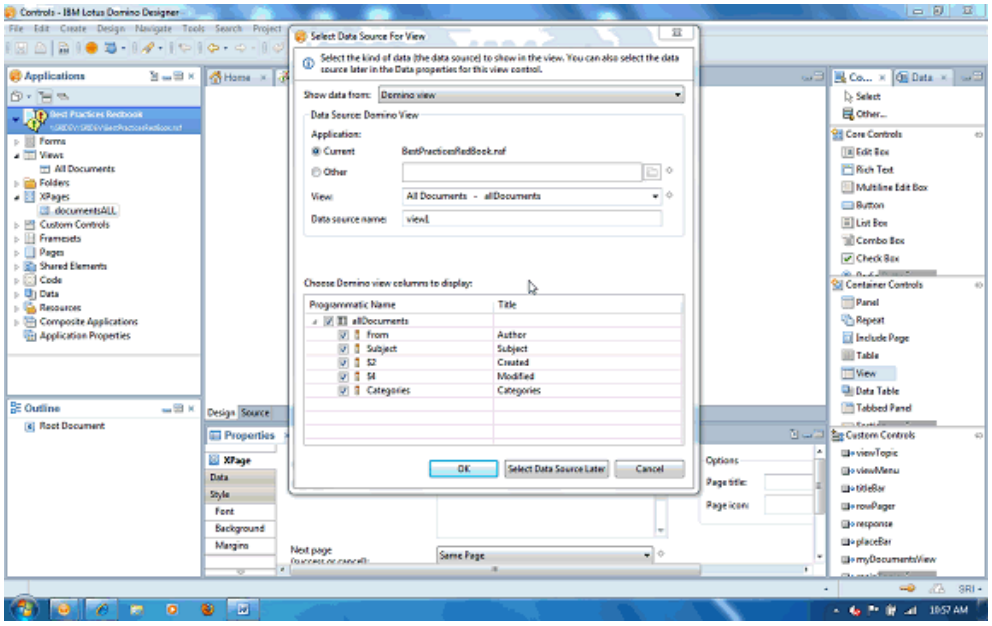
Creating the All Documents view

Earlier, our application contained a view that displayed all documents in the database, sorted by creation date. Here you build the same single view of your data using XPages.

First you create the XPage called **documentsALL.xsp** in Domino Designer, and insert the View Control, as shown:



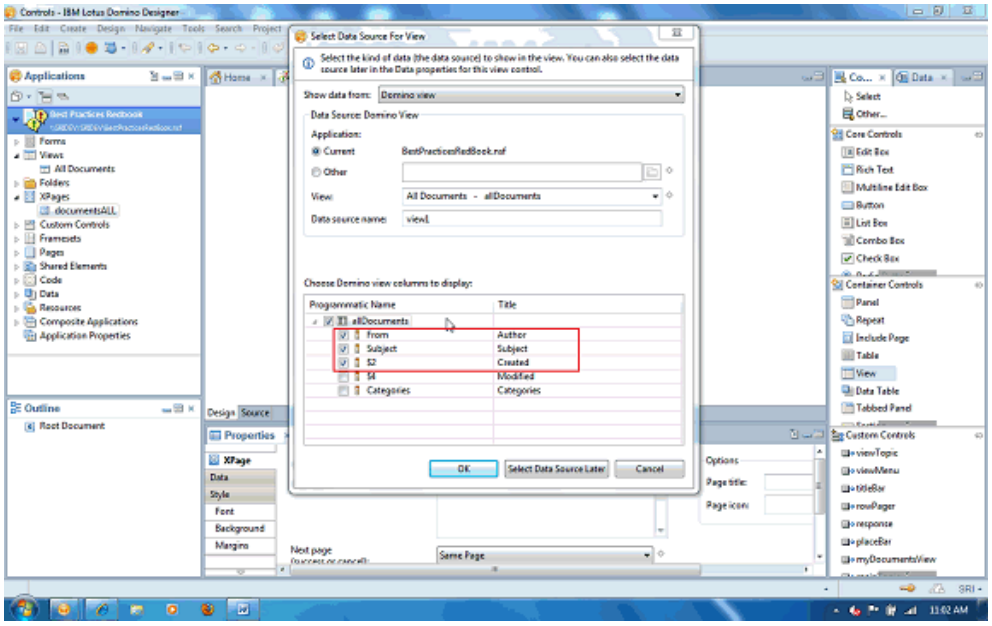
Upon dragging and dropping the View Control onto the XPage, the Select Data Source for View panel shown asking you to complete certain properties:



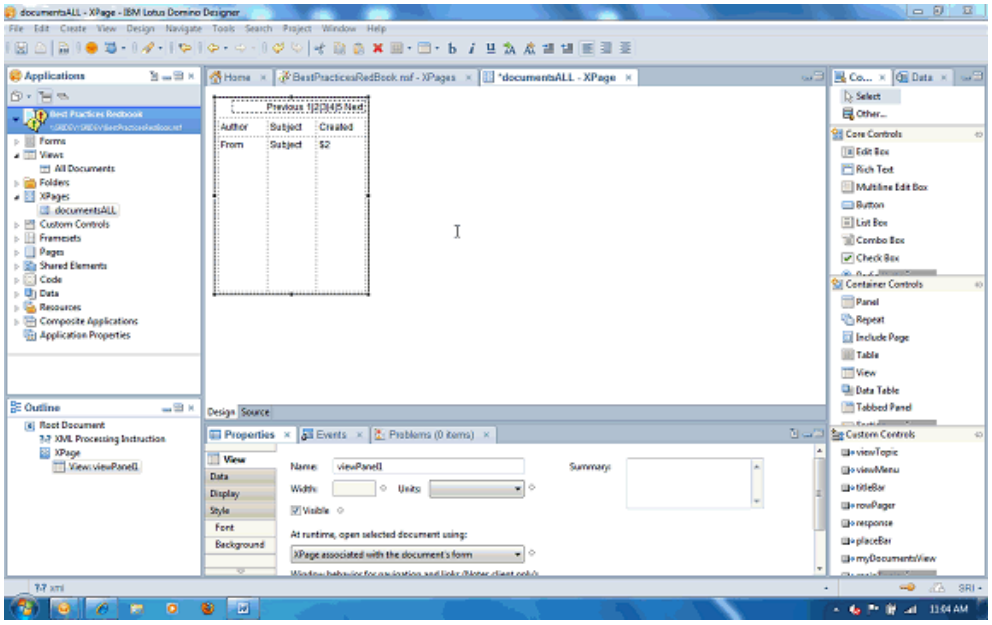
The properties are described below:

- **Show data from**: This field instructs the View Control from where to show data. We use Domino view.
- **Application**: The application for this view. We use our demo application. You can link this View Controller to a view in another Domino application**.**
- **View**: The name of the view to display in this View Controller (the All Documents view)
- **Choose Domino view columns to display:** This is where the power of the View Controller comes in to play. You can select which columns from the view that you want to see, ignoring those that you do not.

Since the original All Documents view contained the creation date, author, and subject fields, we include only those in our XPages design for the All Documents View Controller, as shown:



Your XPage appears as follows, where you can "tweak" the results to your needs.

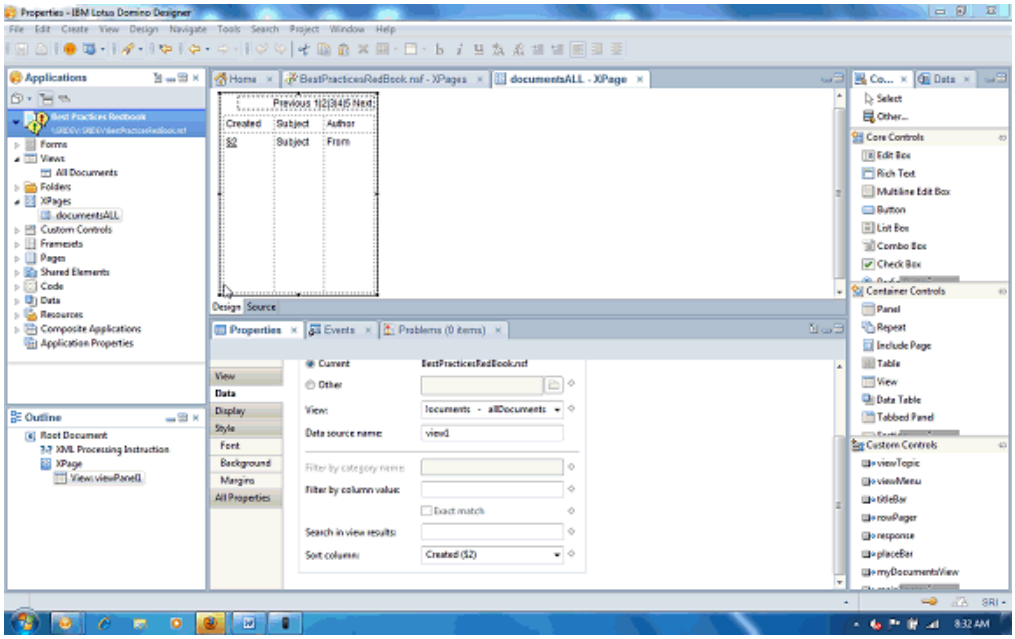


Through the View Controller, you can control virtually all aspects of the view display, including:

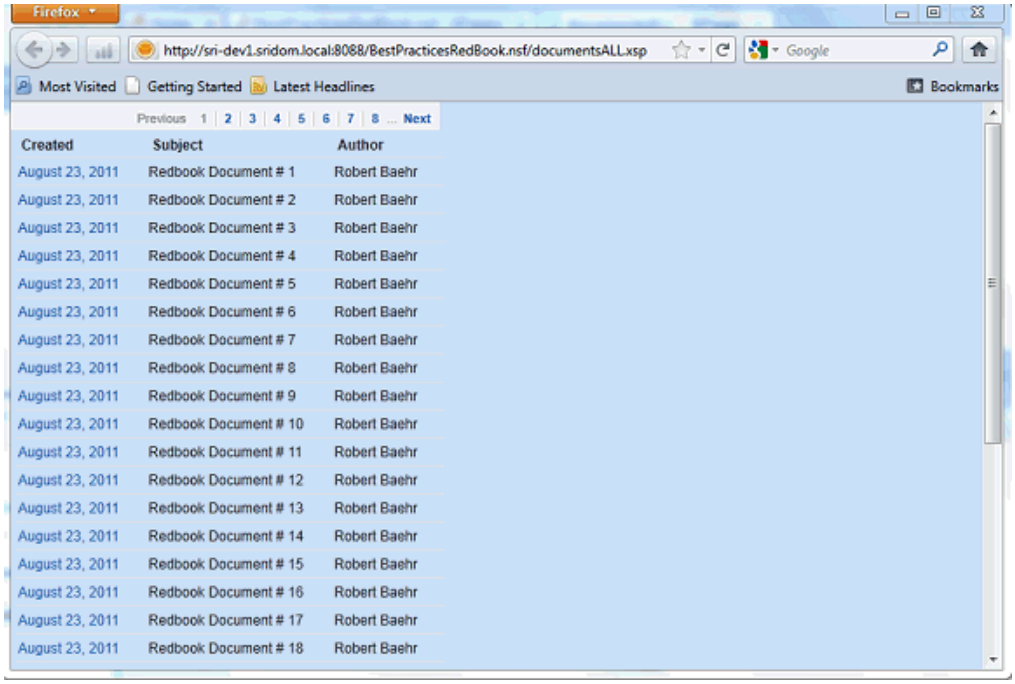
- The navigation bar at the top allows the user to page through the documents in a clean, easy to use navigation system.
- The view columns can be rearranged to suit your needs.
- The view sorting options can be set here on the XPage – a more efficient method of processing view column sorting than traditional Domino views.

NOTE: XPages are not magic; the work of sorting still has to be done at some point. For example, you can rearrange our columns, as needed, by simple cut and paste

operations. Then, you can open the properties of the Created column and instruct XPages to sort the data, as shown:



Upon preview, you obtain the following XPage-based view:



Created	Subject	Author
August 23, 2011	Redbook Document # 1	Robert Baehr
August 23, 2011	Redbook Document # 2	Robert Baehr
August 23, 2011	Redbook Document # 3	Robert Baehr
August 23, 2011	Redbook Document # 4	Robert Baehr
August 23, 2011	Redbook Document # 5	Robert Baehr
August 23, 2011	Redbook Document # 6	Robert Baehr
August 23, 2011	Redbook Document # 7	Robert Baehr
August 23, 2011	Redbook Document # 8	Robert Baehr
August 23, 2011	Redbook Document # 9	Robert Baehr
August 23, 2011	Redbook Document # 10	Robert Baehr
August 23, 2011	Redbook Document # 11	Robert Baehr
August 23, 2011	Redbook Document # 12	Robert Baehr
August 23, 2011	Redbook Document # 13	Robert Baehr
August 23, 2011	Redbook Document # 14	Robert Baehr
August 23, 2011	Redbook Document # 15	Robert Baehr
August 23, 2011	Redbook Document # 16	Robert Baehr
August 23, 2011	Redbook Document # 17	Robert Baehr
August 23, 2011	Redbook Document # 18	Robert Baehr

Note that we did not make the interface attractive, for our focus is on view design and performance.

In summary, here is what XPages design brings to views, view performance, and server resource utilization:

- We created one view in Domino Designer, thus, Domino has one index to maintain.
- We inserted the view onto our XPage, using the XPage View Control, removed the columns we do not want, rearranged the column we do, and sorted on the appropriate column – giving us the custom view we need without physically designing another view in the application.
- By selecting different sets of columns to display, we can create different front-ends to the same Notes view. To display different selections of documents, however, you usually still need different views.

To illustrate the last point, consider building an XPages-based view for the following views:

- documentsAuthor.xsp: All documents, sorted and categorized by the Author, and containing the Subject, Creation, and Last Modified fields
- documentsCreated.xsp: All documents, sorted and categorized by the Creation date, and containing the Subject, Author, and Last Modified fields

Using traditional Domino development, you would have to design two additional views, so the server would have to maintain two additional view indexes. Using XPages, you can repeat the process above, inserting only the columns you need, giving users three views from one physical view design.
XPages interfaces have drawbacks as well as advantages.

- It's somewhat more work to design.
- Full-text search and other functions that users take for granted in a Notes view, are not available unless you design them in each time.
- It's difficult to navigate longer views, where you may need to advance to the next page many times to get into the middle of the view. The Notes view's "quick search" function and the scroll bar that can jump you to any place in the view, are a big help when working with large amounts of data.
- There is no way to do a manual full-text search on the whole view, select all matching documents, and execute an action on them, if the results are more than one page long. This is routine in a Notes view.

6.1.9 Summary of the view performance best practices

- Limit the number of views.
- Limit the complexity of views -- fewer columns, simpler formulas.
- Avoid using @Now, @Today, etcetera in view selection and column formulas.
- Take time to consider whether you have set the appropriate indexing options for each view.
- Limit use of user-sortable columns, while bearing in mind that a single re-sortable view has less performance impact than two views with no re-sort.
- Don't allow end-user creation of shared views. The users don't understand the performance impact of creating views, and may create many duplicates.
- Avoid excessive column categorization in the design. A categorized column affects performance significantly more than a simple sorted column.
- When categorizing multiple columns in a single view, it is best to have a single categorized column, using the value "Column1Value\Column2Value\etc". Because Domino only has to maintain one category index. [really?]

- When displaying columns that are categorized, or responses that appear beneath their main document, use the view design option "Collapse View When Database Is First Opened" to make views open faster.
- Using XPages and the View Control can enhance the application performance by re-using the same view in different display or sort orders, minimizing index maintenance and size in the application.
- Work with your Domino Administrator to enable Transaction Logging on the server to take the advantage of view index logging (a property of the design of each view) to allow server restart and application availability to be optimal in the event of a server crash.

The main consideration in designing views, is the tradeoff between giving the user complete information, and performance of the application and the server.

As a designer, the Rapid Application Development (RAD) nature of Notes permits you to add a view with ease. While at times this may be necessary, you must really think about simply adding additional views “on the fly” without considering the impact on the overall application, and, the overall environment.

First learn what tasks users need to do in your application, and think through the application design. Design each view carefully to give users the information they need for their specific tasks. Adding unneeded columns and re-sorts hurts performance without helping the user. Unnecessary columns can even make your application more difficult to use by cluttering the screen with trivia.

It's quite common to see applications with dozens of views, many of which are probably unused. Nobody, not the developer, not any one user, knows what all these views are for or which documents they select. The view name in the outline is often not enough information for the user to determine what view to use. Even if they go to a given view, they may not be able to tell whether it contains all and only those documents they need for their task. This sometimes results in users requesting creation of new views that duplicate existing views, simply because the name of the existing view doesn't suggest the task they want to do there.

This can be addressed in part by intelligent design of the navigation area of your application. You don't have room in the outline to describe the view in detail. What you can do instead is identify the tasks that different users do in your application, and create views that directly address those tasks. Sometimes (often, we hope) different groups of users may find the same views useful. Bear in mind that the label of a view in the outline doesn't have to be the same as the name of the view, and that multiple outline entries might point to the same view. Also consider that certain tasks may find it useful to see multiple views at the same time as a document preview. Perhaps a frameset or composite application would be helpful. Sometimes, views are identical but for different sets of actions, directed to different tasks. Consider merging these views and using action hide formulas based on @UserRoles instead.

If you learn what tasks your users do in your application, and design around those tasks, you will maximize productivity while also keeping the number of views under control and being able to tell when a view is obsolete, so that you can delete it and improve overall performance. If you already have an outline that isn't task-oriented and you want to let people find views in the old way also, consider a tabbed UI in the navigation space that would let users select the "legacy" navigation or the new task navigation.

There's no way to directly test whether a given view is unused. As we saw in some earlier examples, you can view the index information of specific views and tell whether the index

has been deleted for disuse. However, because Notes data are replicated but view indexes aren't, this information may be different on different servers. Also, especially in applications with a large number and confusing organization of views, users may open the wrong view by mistake. So the existence of an index is no guarantee that the view is needed. If you think a view might be unused and you'd like to delete it, you might add code to the Postopen view event to display a dialog that the view is to be deleted on such and such a date if you don't hear from anyone who wants to keep it. Or even delete the view and change the outline entry to point to a page that says "this view has been deleted because we didn't think anyone was using it. If you want it back send mail to ___ explaining why you need it. Or, have a look at this other view that we think fulfills the same purpose: (link here)." If you don't hear from anyone for a while, delete the outline entry also.

6.2. Collections

In your LotusScript or Java code (including SSJS), you build collections for processing. There are performance considerations to evaluate both in building those collections and in processing the collections.

6.2.1 Building collections

There are several methods used to build collections, and each have their pros and cons. The methods below are applicable to LotusScript code and Java code (including server-side JavaScript).

NotesDatabase search

Inside the NotesDatabase class (either LotusScript or Java), you can call the search method to build a collection. This method takes a search string (which is in the form of a SELECT statement, similar to a view selection formula), a cut-off date (which can be the keyword Nothing in LotusScript or null in Java), and a maximum number of documents to return (which can be 0 to return all documents).
The NotesDatabase.search method is generally *not* the best choice when building collections. This method doesn't take advantage of existing view indexes or the full text index. It must open every document in the application to determine whether that document matches the criterion.

It can be efficient if you use the date/time argument to limit the search only to very recent documents, because the NSF maintains an internal index of documents sorted by when modified. But in general, other methods that search a view index or full-text index to locate documents of interest, will give better performance.

NotesDatabase UnprocessedDocuments

In scheduled agents, using the UnprocessedDocuments method is a fast way to get the documents that have been modified since the last time the agent ran. You can think of each agent having its own list of “read” and “unread” documents. If your agent only has to process new and modified documents, using this method will return the best performance.

NotesDatabase FTSearch

If your application has a full text index, using the FTSearch method searchs for documents to process with adequate performance. However, if the application does not have a full text index, this method performs poorly because an entire full text index must be built in memory, then searched, then discarded. You may wish to use the NotesDatabase.IsFTIndexed property and NotesDatabase.UpdateFTIndex to make certain the index exists and is up to date before you try to use this method. Creating the index when needed using the CreateFTIndex method, only works for local databases.

One drawback in using the FTSearch method is that there is no way to search for exact equality. For example, a search of [Subject] = “Test” will find documents where the subject field is the value “Test”, but also find documents where “Test” is a part the subject field (such as "Field test status").

Note that the FTSearch method can also be used against a NotesView object to perform the full text search against just the documents contained in the view. However, it is generally slower than a search of the whole database. This is because the full-text index is maintained against the whole database, not against individual views. So to do a full-text search of a view, Notes must first search the whole database, then iterate through the results to see which of those are in the view.

NotesDatabase AllDocuments

This method loads into memory all the data in the whole application. If the application has a large number of documents, this is a very memory intensive method. It would be more efficient in this case to use a view that contains all the documents. However, this method is more efficient than performing a search that results in all the documents in the application.

NotesDatabase FTSearchRange

The FTSearchRange method is similar to the FTSearch method with one additional parameter (the starting document to return). The best practices applicable to FTSearch are also applicable to FTSearchRange.

NotesDatabase GetAllUnreadDocuments

The GetAllUnreadDocuments obtains all the documents that have not been read for the current user (or, optionally using a parameter, the user name passed in as a parameter). For your scheduled agents, this is similar to the UnprocessedDocuments method. However, if the application is not tracking unread marks, all documents are considered read. If unread marks are not tracked, the UnprocessedDocuments method is a better choice.

Note that there is also a GetAllReadDocuments methods that returns documents that have been marked read for the current user or the optional parameter user name.

NotesDatabase GetModifiedDocuments

This method is used to return a collection of documents that have been modified since a specific date and time. This method performs well because of the internal structure of Domino applications.

NotesDatabase UnprocessedFTSearch

This method first obtains the unprocessed documents in the application, and then performs a full text search against the collection. There is also an UnprocessedFTSearchRange method that includes a starting document to return. And there is an UnprocessedSearch method that is similar to the Search method, except the collection to search is the unprocessed documents instead of all documents in the application.

These methods perform slightly worse than their comparable methods because of the additional overhead of getting the collection of unprocessed documents first.

NotesView GetAllDocumentsByKey and GetDocumentByKey

If you have a NotesView object that has one or more stored columns (a categorized column is also a sorted column), you can get a collection documents that match values in the sorted columns.
The GetAllDocumentsByKey method itself is a good performing method, but a view must be in memory first. The GetView method in the NotesDatabase object is used to read that view index into memory before GetAllDocumentsByKey can be used. The GetView method is an expensive method from a performance standpoint. So, even though GetAllDocumentsByKey performs well, you have to factor in the expensive GetView method when comparing methods of getting a collection of documents.
These methods are generally the most efficient way to locate documents by key values, and it is highly recommended for performance reasons, especially when you have to do repeated lookups using the same key (because you only have to call GetView once). These methods can take an array as the key argument, locating only documents that match all the key values (corresponding to the first however many sorted columns are in the view).

If the user has re-sorted the view, it does not affect this method. Note, however, that you can use the NotesView.ResortView method to select which column you would like to search by, if it is not the default sort column of the view. Note, however, that it's generally considered a best practice to not do your searches in user-visible views, since these tend to get changed, and that might affect your code.

NotesDocument Responses

If you have a document in an application that enables specialized response hierarchy information, this property of the NotesDocument object is a fast way to get all the immediate responses to the current document.

LotusScript only methods for getting collections

There are a few LotusScript only methods used to get collections. These are from the LotusScript front-end classes and have no Java equivalent. For more information about these functions, see the Domino Designer help documentation. The methods are:

- NotesUIWorkspace.PickListCollection
- NotesUIDatabase.Documents
- NotesUIView.Documents

6.2.2 Processing collections

Once you have a collection inside your code, you have to process the documents in the collection. You can process the documents iteratively or sequentially.

Processing collections sequentially

To process a collection sequentially, use the GetNthDocument method.

Important: The GetNthDocument method should rarely, if ever, be used. The GetNthDocument method uses the first document in a collection as a reference point every time it is called. It then steps through all the documents until it reaches the Nth document. For example, if the code is trying to find document number 900 in the collection, it starts with document 1 and then counts through the rest of the collection until it finds document 900. If you have to loop through a number of documents, you will experience better performance processing the collection iteratively.

Processing collections iteratively

To process a collection iteratively, you either start with the first document of the collection and then process the next document, or you start with the last document of the collection and then process the previous document. The methods used are:

- GetFirstDocument and GetNextDocument
- GetLastDocument and GetPrevDocument

In LotusScript, the GetNextDocument and GetPrevDocument methods take a parameter of the reference document in the collection. In Java and SSJS, the preferred method is to omit the parameter and loop through the documents. The example code loops through all the documents in the database:

```
try {
     DocumentCollection dc = null;
     // You must create the collection before this code will actually work
     Document tmpdoc;
     Document doc = dc.getFirstDocument();
     while (doc != null) {
          // Document processing goes here
          tmpdoc = dc.getNextDocument();
          doc.recycle();
          doc = tmpdoc;
     }
} catch(Exception e) {
     e.printStackTrace();
}
```

6.2.3 Manipulating collections

In addition to processing collections, you can manipulate collections by adding or removing documents from the collection. To add a document to the collection, use the AddDocument method. To remove a document from the collection, use the DeleteDocument method. There are also methods Merge, Intersect, and Subtract which are used to manipulate two collections and return a brand new collection.

Note that there is almost no difference, from a performance standpoint, between the methods to manipulate collections. For example, if you have a document and you want to add it to a collection, you can use either the AddDocument method or the Merge method to manipulate the collection.

6.2.4 Additional collection objects

In addition to the NotesDocumentCollection object, there are two other objects which handle collections of documents.

NotesNoteCollection object

The NotesNoteCollection object is similar to a NotesDocumentCollection object, with the exception that the NotesNoteCollection object can contain design elements in addition to documents. There are Add, Intersect, and Remove methods to manipulate the contents of the collection. Performance with this object is similar to the NotesDocumentCollection object.

NotesViewEntryCollection object

Inside the NotesView class, there is the AllEntries property and the GetAllEntriesByKey method that return a NotesViewEntryCollection object. This object contains documents, just like other collections discussed in this article. However, the documents in these collections are sorted based on the view criteria. If your code needs sorted documents, it will generally be faster to use a NotesViewEntryCollection object rather than trying to sort a NotesDocumentCollection or NotesNoteCollection in your code.
Keep in mind that, since you are accessing data through a view, it is a best practice to set the AutoUpdate property in the view to False to get the best performance. This property prevents the code from checking to see if the index in memory is out of date and updating the index if there was a change. If you have to ensure the latest index in your code, then use the Refresh method in the NotesView object at the appropriate time to get the latest index.

6.3. Documenting data access

Once you have a document (NotesDocument in LotusScript or Document in Java or server-side JavaScript), you might need to read field values off that document. This section describes the best practices for accessing data off a document.

6.3.1 LotusScript methods

There are two ways to access data from a document in LotusScript. The first is through something called the Extended Syntax and the second is through the GetItemValue method in the NotesDocument class. From a performance standpoint, both methods take a similar amount of time. However, there might be other reasons for choosing one over the other.

Extended syntax

The extended syntax is a way to directly access a field by its name. For example, to read the value of the “form” field off a document, the following statement uses the extended syntax:

```
result = doc.Form(0)
```

GetItemValue method

The GetItemValue method takes a parameter of the field name. It returns an array of values where the data type of each value depends on the data type stored in the document. Many developers will quickly access the first value of the array in their code. For example, the two code blocks below return the first value of the “Form” field off the document:

```
result = doc.GetItemValue("Form")(0)
```

and

```
resultArray = doc.GetItemValue("Form")
result = resultArray(0)
```

Comparing extended syntax to GetItemValue

The performance comparing GetItemValue and the Extended Syntax is nearly identical. Choosing which method to use comes mainly down to personal preference.
The major advantage to using the extended syntax is the number of keystrokes required to enter in the statement. Although, with type-ahead available in Domino Designer, the GetItemValue method can be automatically completed for you in a few key strokes.
One disadvantage to using the extended syntax is using a field name that might, in a future release, become a method or property name. For example, the MarkRead method was introduced in Domino 8 Designer. If your code previously accessed a field called MarkRead with the extended syntax, that would no longer work on a Domino 8 client or server.
One advantage to using the GetItemValue method is portability to Java.

6.3.2 Java methodsJava methods

Inside Java and server-side JavaScript , the Document class has several ways of reading data, depending on the type of data being read. The getItemValue method returns a vector containing all the elements. The getItemValueDateTimeArray method is used to read date time values or a single date time value. However, if there is only a single non-date time value in the field, quicker access to the value is through one of the other methods listed below:

- getItemValueDouble
- getItemValueInteger
- getItemValueString

6.3.3 Avoid Opening Notes Unnecessarily

When processing documents obtained from a view (either by a key search or by iterating through the view) it is more efficient to access information about the document using the NotesDocument.Columnvalues or NotesViewEntry.Columnvalues property rather than referring to a field directly. Column information can be obtained from the stored view index, without having to access the note data, so it is faster.

6.4. Readers field and view performance

As one of the most used document-level security features, the Readers fields allows you unparalleled document (record) level security in the Domino applications. This section offers a review of the Readers field and how the Readers field secures Domino data, some of the performance pitfalls that you might encounter if you use them in conjunction with poor view design, and how you can overcome these performance limitations.

When define a field on a form with the data type “Readers”, you can enter one or more canonical Notes names into the field, thus, restricting whether the view returns the document for display and subsequent reading. You are not limited to one Readers field on a form. If you deploy more than one Readers field on a single form, Domino aggregates these fields into one “Master” reader list. Any name that appears in an Authors field, also has read access to the document.

The names may be individual user or server names, role names, or group names (of groups defined in the Domino Directory). It's considered a best practice, to avoid hardcoding user or group names into your application. Doing so would prevent reuse of the application by a different group of users, and would complicate maintenance. The ADMINP process can find and update names in documents, when users or groups are renamed. But if the name that's out of date is in a formula in a computed field, ADMINP can't fix it. Use roles, or else store names in configuration documents, so they can be managed by the servers.

Note we are *not* saying that you shouldn't store user and group names in these fields. Merely that the names should not be part of a formula or other code in your application. Letting the user select names from the Domino Directory, or using the @Username function in a formula of a Readers field, are perfectly fine.

The following is an example of using more than one Readers field on a form:

- The document is normally considered private for the original author, so a computed-when-composed Readers field has the formula **@Username**, allowing the user to see their own document.
- An editable Readers field on the same form lets the user select one or more names from the Domino directory, of people they want to share the information with.
- An Authors computed field has a formula based on the Status field of the document, which supports a workflow process by calculating the name of the next approvers. An Authors field is used here because the approvers must have access to modify the document, to approve it.
- To insure that the servers and server administrators have access to all documents, a computed-when-composed Authors field contains the formula "[Admin]". [Admin] is a role name which you would have to define in the database ACL. It makes sense to

use an Authors field for this as a general rule, rather than a Readers field, because often there are cases of optional Readers fields -- the field may be left blank, allowing access for all, but if you also have a non-blank "admins" Readers fields, then *only* the administrators will have access.

Once saved to the application, when a user accesses the document through a view, the document will not be visible to the user if his canonical name is not in one of the Readers fields. This is true even for documents created by that user.

Note: If you configure **Full Access Administration** on the Domino Server document, and open a database through the Domino Administrator with Full Access Administration enabled, you can view and edit any document in the application, regardless of Readers fields. For more information, see **Domino Administration Help**, and search on "full access administration".

6.4.1 View and Readers fields

To understand the impact of Readers fields on performance, you have to think about how documents are stored in a view and how the view is displayed to users. In summary:

- For a server view, the view index is shared among all users. The server creates or updates the index without knowing who will use the view next.
- When a user opens the view, if the documents contain Readers fields, the server has to decide which of the documents this user has access to view. So Domino scans the view index to create a subset of documents and associated column values to display for that user. We call this the *working set*.
    - When encountering a document that contains one or more Readers fields, Domino compares the values in those fields and any Author fields, against the list of names, roles and group names associated with the current user. If there is a match, the document is added to the working set.
    - Also, of course, any documents that don't contain Readers fields are added to the working set.
    - Once the working set is computed, it must be refreshed again before it is displayed. (?)

It is in Domino's use of the special filter and post-filter indexing operations that, depending on the use of Readers fields, the number of documents to be displayed in the view, and the column data used in the view design, there might be a negative performance impact when opening a view. The more documents that have Readers fields, the more work must be done to determine which ones the current user has access to. And, the more values there are stored in the Readers fields, and the more groups the user belongs to, the longer each individual document test will take.

6.4.2 Best practices in using reader fields effectively

To design views to display documents using Reader fields for optimal performance, consider the following techniques:

- Categorized columns
- Embed the view on a Page or Form using the single category option
- Local replicas

Categorized columns
If you have views in you applications that display documents utilizing Readers fields, consider categorizing the view and enabling the "Collapse View When Database Is First Opened" view design property. This allows Domino to more quickly process the documents to display in the view (because Domino only has to initially show the column containing the category, not multiple fields from multiple documents).

The consideration for taking this approach is that the user must then expand the category to obtain access to the data (and the document), which can take time depending on the number of documents in the category since the Readers fields in the category's documents must be evaluated at that time. But this is faster than evaluating all the documents in the view.

Of course, this does mean that there may be category headings displayed, which don't contain any documents that the user has access to. Depending on the application and on which columns are categorized, this may be considered a security exposure, because the user can tell that there are documents containing the displayed value, even though they can't see the documents themselves. There's an option to hide categories that contain no documents the user has access to, but then you lose the performance advantage.

Embed the view on a Page or Form using the single category option

If the user mainly has read access only to "their own" documents, identified by their name stored in a field, you can give them a "my documents" view by categorizing the view by the field that contains the usernames. Embed this view on to a Page and use @Username or @v3Username in the "Single Category" formula of the embedded view to select only the category containing the user's documents.

The benefit of this technique is that the view will display quickly, for Domino only has to show documents belonging to the user (the single category), and doesn't have to consider whether the user has access to documents in all the other categories.

The consideration of this technique is that application maintenance will be greater because you lose “native” access to the view, thus, must develop the page to provide the functionality you need (for example, expanding and collapsing the view).

Local replicas

Another technique to optimize view performance with respect to Readers fields is to maintain local replicas of applications on your workstation. Replication assures that only the documents that you must have are replicated down to your local workspace, because the server only the sends the client documents which the user has read access to.

For example, if there are 100,000 documents in the server-based application, and, you only have Readers access to 10,000 of those documents, the local replica contains 1/10th of the number of documents that the server-based application has, thus, indexing will be more efficient and view performance will be increased. In addition, local replicas tend to be more efficient even when they don't contain fewer documents.

Considerations for using this technique include replication latency (that is, access to documents may not be in ‘real-time”), and, application maintenance -- the support of multiple application replicas residing on the server and on client workstations.

Direct Access to Document

Another way to avoid view-indexing delays in accessing a user's document, is most appropriate for applications where the typical user has access to just a single document. You don't give the user access to any views; instead, when they open the application, it finds the one document to which they have access, and opens that in a document window. This might also be done by emailing the user an automatically generated memo with a doc link to their document.

6.5. Domino application scalability

The Domino applications (single NSF) are supported up to 64-GB in size on the Windows and UNIX platforms. This section describes certain best practices for scaling the application for maximum accessibility and performance, including various application settings and deployment scenarios.

Fundamental scalability techniques
At the application level, you can use the following techniques to assure maximum application scalability:

- Limit number and complexity of views.
- Avoid unnecessarily modifying documents, and avoid deleting documents and then creating documents exactly like the ones you just deleted.
- Database properties that can impact the overall size (see below).
- Database quotas
- Archive documents
- Selective replication
- Deletion stubs - limit how long they are kept (but bear in mind that this will affect replication)
- User activity logging (slows things down)
- Soft deletions

Advanced scalability techniques
These techniques are typically used for applications that have large numbers of users who rely on the Domino web engine to access the application, and, for applications that may exceed the 64GB limit imposed above:

- Domino clustering
- Internet clustering
- Store the data in an external relational database management system using Domino Enterprise Connection Services (DECS) and Lotus Enterprise Integrator (LEI)

6.5.1 Application compaction

Compaction is a process used by the Domino Server (and Notes Client) to perform many maintenance functions on a Domino application. For the purposes of our discussion, we focus on the Domino Server commands for application compaction, and, discuss the best practices for getting the most out of application compaction.

The compact command is invoked by using the following command syntax:

- From the server console:

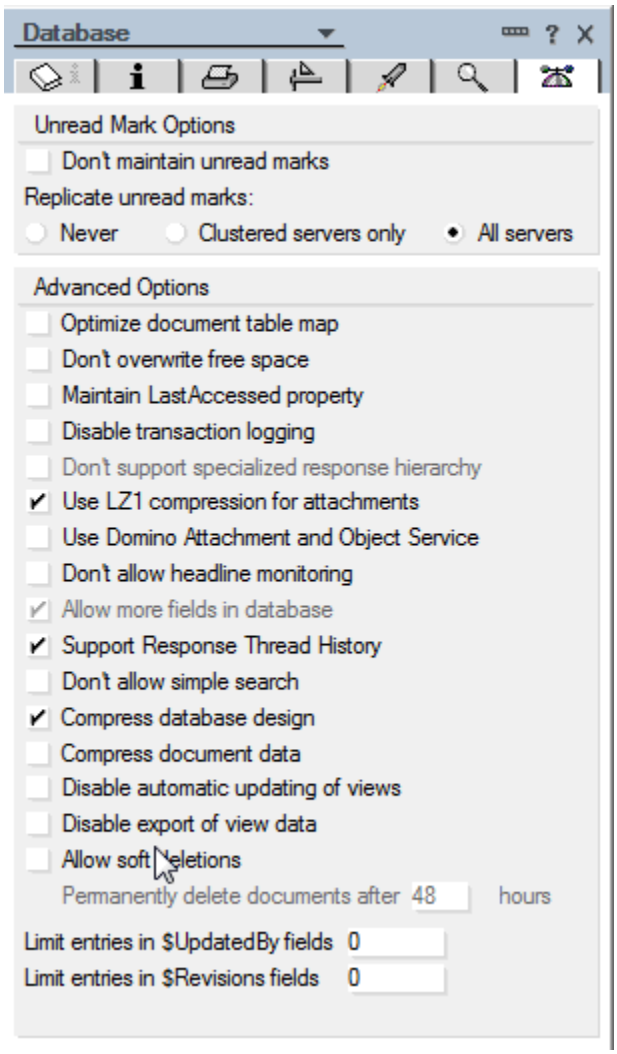
```
LOAD COMPACT ApplicationName.NSF –Options
```

- From a Notes client:
    a. Open a command prompt.
    b. Using the **CD** command, change to the Lotus Notes program directory. For example, **CD "\Program Files\IBM\Lotus\Notes"**as
    c. Run **NCOMPACT ApplicationName.NSF *–Options***

For the complete list of Compact server command line options, see the TechNote # 1084388 at the Lotus Notes Support web site at http://www.lotus.com/support

6.5.2 Application properties for scalability

Each Domino application maintains an independent set of properties that control applications features, and application performance and scalability. You can access these properties from **File -> Application -> Properties** menu, then select the **Advance** icon ( ), as shown:



Given the numerous possible application design scenarios, here we focus our discussion on general best practices for using the relevant properties which impact application scalability and performance. The following table shows a summary of the key application properties, from the Lotus Notes 8.5.3 C5 Beta Help. You must compact the application after enabling sand disabling one or more of these settings.

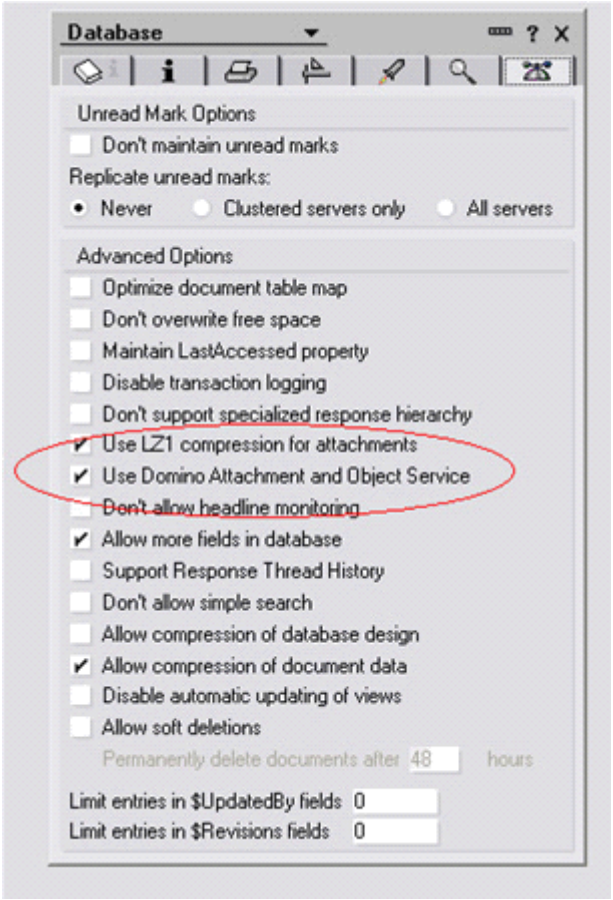
Property	Tab	To optimize performance/size	Improves database performance?	Reduces database size?
Allow use of stored forms in this database	Basics	Deselect option	Yes	Yes
Display images after loading	Basics	Select option	Yes	No
Don't maintain unread marks	Advanced	Select option	Yes	Yes
Optimize document table map	Advanced	Select option	Yes	No
Don't overwrite free space	Advanced	Select option	Yes	No
Maintain LastAccessed property	Advanced	Deselect option	Yes	No
Don't support specialized response hierarchy	Advanced	Select the option	Yes	Slightly
Don't allow headline monitoring	Advanced	Select the option	Prevents performance degradation	No
Limit entries in $UpdatedBy fields	Advanced	Select the option and specify the number of entries $UpdatedBy fields can contain	Yes	Yes
Limit entries in $Revisions fields	Advanced	Select the option and specify a limit on the number of entries $Revisions fields can contain. The suggested limit is 10 entries.	Yes	Yes

6.5.3 On-disk structure level

With each major release of Lotus Domino, IBM adds functional and performance enhancements to the database file. Each version of Domino ships with the same standard database file format, which is referred to as the on-disk structure (ODS). The on-disk structure level of a Domino application is an important consideration in any application deployment.

For example, the Domino Attachment and Object Service (DAOS) is a storage method whereas all file attachments are stored in a single repository and referenced as needed. This saves storage in the NSF and on your Domino server.

To use DAOS, you must have you mail databases set to, at a minimum, ODS 50. If an application is set to a lower ODS level, you cannot access the Domino application property that relates to DAOS, as shown:



To upgrade server-based applications to ODS 51, the latest ODS version for Domino 8.5.3, issue the following console command:

```
LOAD COMPACT -c <databasename.nsf>
```

To have Notes automatically create new databases using ODS 51, add the following parameter to the NOTES.INI file:

```
Create_R85_Databases=1
```

Best practices for on-disk structure management

- Domino applications should always be upgraded to the latest supported ODS level.
- The ODS level does not replicate, therefore, you must plan to upgrade all applications on all machines – client and server.
- For Servers, the ODS level should be set to the highest level supported by that version of Domino.
- For Clients, the ODS level should be set to the highest level supported by that version of the Notes Client.
- To have new Domino applications created using ODS Level 51, enter the Create_R85_Databases=1 parameter in the NOTES.INI file.
- To upgrade existing Domino applications to ODS 51, issue the console command **LOAD COMPACT –c** .

6.5.4 Attachment handling

As a true rich information store, a Domino application can handle a variety of data (text, numbers, graphics, and attachments) quickly and easily. From a usability standpoint, attachments can degrade system and application performance, unless care is taken to properly handle applications containing many attachments.

Attachment considerations

If your applications require extensive use of file attachments (such as spreadsheets, documents, presentations, PDF files, and so on), you must carefully look at various settings that can impact the performance of both the application and the system as a whole.

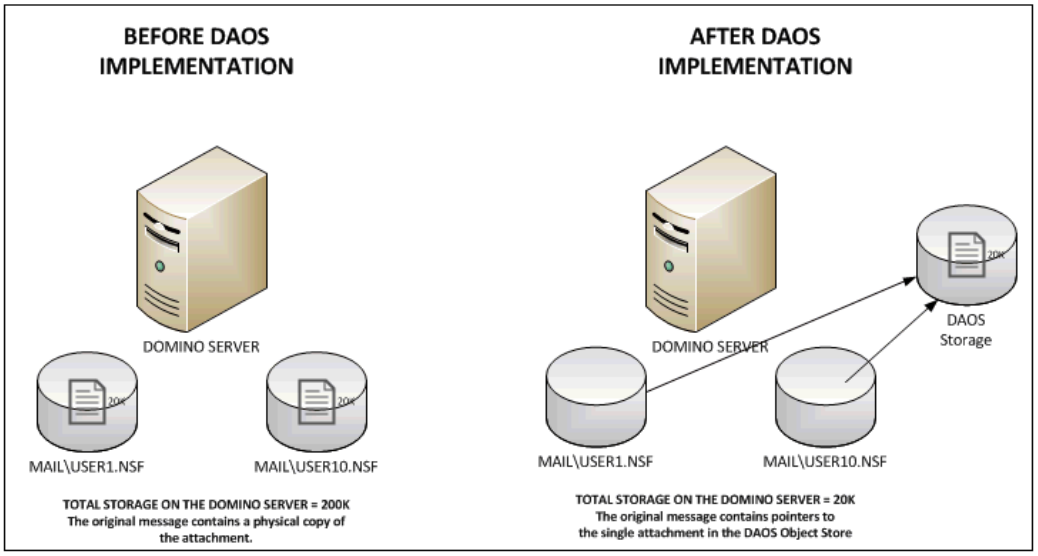
These considerations include the number of attachments, the types of attachments, search options, and methods that you can deploy to help Domino more efficiently handle these operations on your application.

Attachment consideration: Number of attachments

As the number of attachments grows, our application must manage and maintain them. In the case of electronic mail, this problem becomes even more cumbersome.

If you e-mail a 20K document to 10 users, as a messaging system, Domino by default, stores ten copies of that document on the Domino server (one for each recipient). Therefore, the 20K document has ballooned to 200K of storage for Domino to manage. While this example may seem trivial, consider multiplying this over an average number of e-mails per day, and, the average number of recipients in an organization.

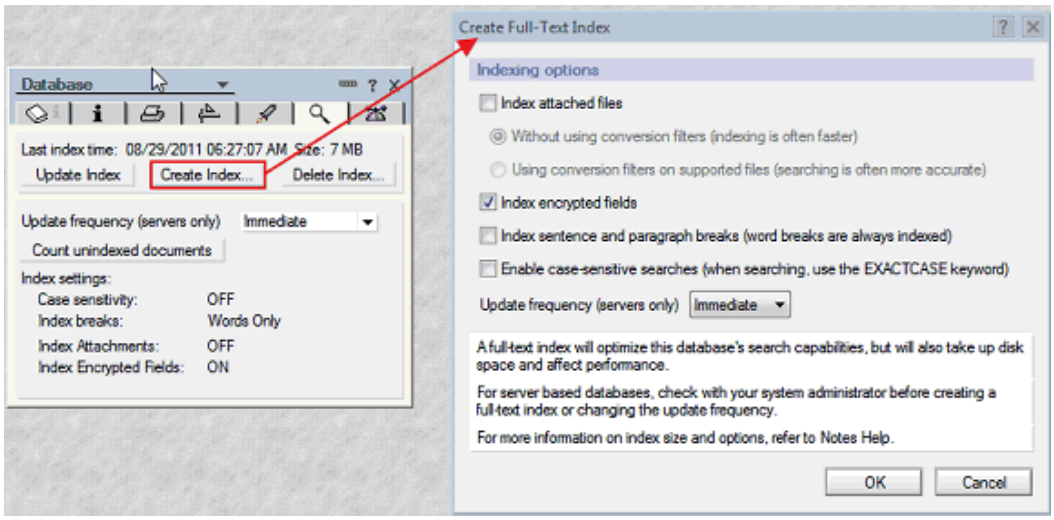
To help solve this issue, Domino 8.5 allows you to enable the Domino Attachment and Object Service (DAOS). This service, when configured properly, can eliminate the problem outlined in the previous scenario by allowing Domino to store only one copy of the attachment in a separate repository. The functionality is best shown by example:


BEFORE DAOS IMPLEMENTATION
DOMINO SERVER
MAIL\USER1.NSF
MAIL\USER10.NSF
TOTAL STORAGE ON THE DOMINO SERVER = 200K
The original message contains a physical copy of the attachment.
AFTER DAOS IMPLEMENTATION
DOMINO SERVER
DAOS Storage
MAIL\USER1.NSF
MAIL\USER10.NSF
TOTAL STORAGE ON THE DOMINO SERVER = 20K
The original message contains pointers to the single attachment in the DAOS Object Store

Attachment consideration: Full-text indexing

As a complete search solution, the full-text indexing is not only index the documents but the attachment content as well, causing excessive use of storage to store the full-text index (located, by default, in the sale directory as the application, but stored in a separate folder named databaseName.ft folder). This *fit* folder contains the full-text index that Domino uses to search for information, and can quickly balloon to more than the size of the actual application itself.

As developers, you must decide whether you want Domino to index the file attachments. This can be controlled using the Domino application properties box, as shown:


Database
Last index time: 08/29/2011 06:27:07 AM Size: 7 MB
Update Index
Create Index...
Delete Index...
Update frequency (servers only)
Immediate
Count unindexed documents
Index settings:
Case sensitivity: OFF
Index breaks: Words Only
Index Attachments: OFF
Index Encrypted Fields: ON
Create Full-Text Index
Indexing options
Index attached files
Without using conversion filters (indexing is often faster)
Using conversion filters on supported files (searching is often more accurate)
Index encrypted fields
Index sentence and paragraph breaks (word breaks are always indexed)
Enable case-sensitive searches (when searching, use the EXACTCASE keyword)
Update frequency (servers only)
Immediate
A full-text index will optimize this database's search capabilities, but will also take up disk space and affect performance.
For server based databases, check with your system administrator before creating a full-text index or changing the update frequency.
For more information on index size and options, refer to Notes Help.
OK
Cancel

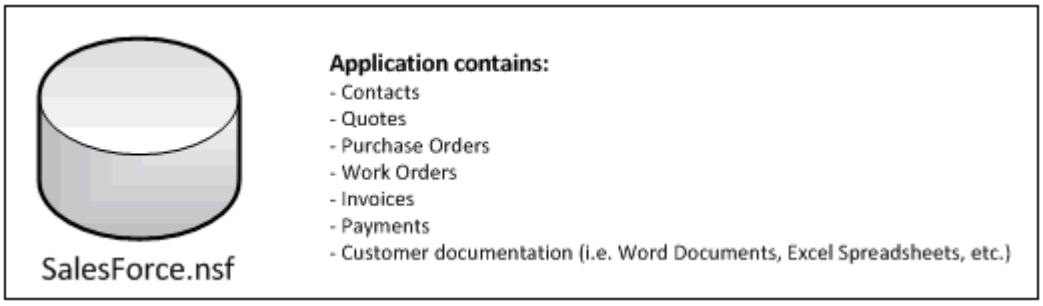
You can tell Domino whether you want to index attachments. The benefit of indexing the attachment is that search results will be more accurate because you are searching the document and the physical attached file. The considerations include the size of the index and the increase in time that searching will take (because of the additional attachment information that has to be indexed), and, that index management time will increase due to Domino's need to index the Notes document, and, the attached files.

Tip: If you do not want to index the attachment, but, still need to search the textual contents of the attachment, you can contact the application vendor to see if they have an application programming interface (API) to access the contents of the attachment. You could then write an agent (possibly in the QuerySave event of the form) that would extract the textual data from the attachment and store it in a hidden field within the document. Then, when you search the document, you are also searching the attachment text, but, without the excess performance hit of actually indexing the physical attachment. The trade-off being that the application increases in complexity, and that you are storing the textual information from the attachment twice – once in the physical attachment, and again in the Notes document.

6.5.5 Multi-database applications

While it would be optimal to have a Domino application comprise of a single storage facility, this may not be possible due to the complexity of the application, the number of documents, attachments, indexing, and design maintenance issues that can arise.

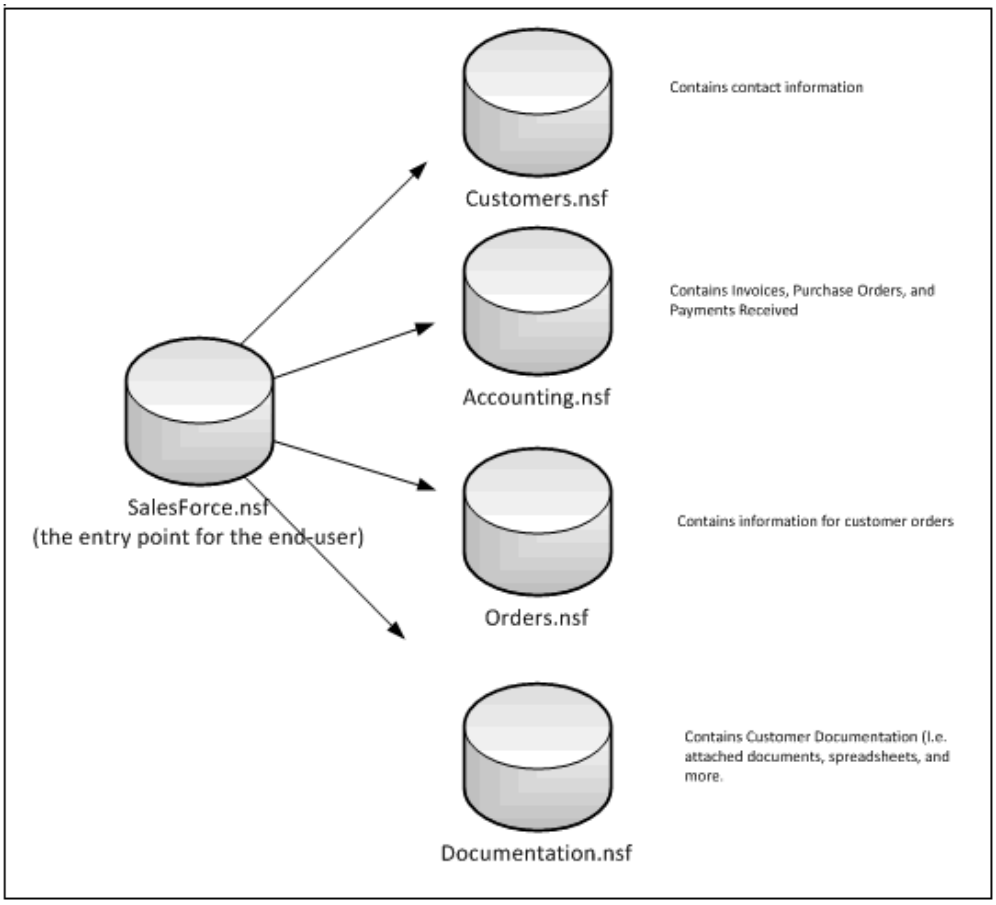
In instances where an application simply gets too *big* for you (or Domino) to efficiently manage and maintain, you have the option to break the application into separate database files, working in tandem to form a complete, cohesive application experience. The benefit of a multiple-database application is that information can be more efficiently processed by Domino. The consideration is that application integration from a Domino developer's viewpoint can become more complex. As an example, consider a Sales Force application, defined as follows:



While this might appear to be a trivial application, consider the performance of the application with respect to:

- The number of views per data item (contacts, quotes, and so on)
- The number of attachments for customer documentation
- The overall number of documents housed in this application

From a performance and scalability perspective, it might be feasible to design the application using multiple data stores, as follows:



Some considerations when building multiple-database applications include:

- **"Relational" database management:**
  In a multiple-database application, it is not uncommon for a document in one database to be related (or linked) to another document in another application. While Domino is not a Relational Database Management System (RDBMS) in the traditional sense, it is possible to link documents together using the Document identifier (DocumentID) uniquely assigned to each document or another unique value designed by the application developer.
- **Application complexity and maintenance:**
  The application complexity and design maintenance will be more difficult because cross database references are required to be considered.
- **Searching:**
  To effectively search for information, you have to implement a multi-database search (for example, implementing multi-database search in Lotus Domino)

Overall, and over time, the application performance will improve in the multi-database implementation. Your job as a developer will be more challenging, but, if you consider the performance and scalability advantages of a multi-database implementation of our applications, the benefits can far outweigh other factors.

6.6. Administration considerations

Although this guide is indirected at the Domino Developer, knowing basic administrator skills is beneficiary in developing applications. After all, a developer can directly impact the performance of the server. This section describes certain settings that, although controlled by the administrator, might need adjustment depending on the intended use of the server from a development standpoint.

6.6.1 Java heap size

In Domino 8.5, a notes.ini parameter was automatically added to the Domino server:

```
HTTPJVMMaxHeapSize=256M
```

This parameter was added to control memory allocation to allow the new XPages functionality to work correctly. Previously, the **maximum** heap size was 64 MB. Starting in Domino 8.5.2, the **default** value for the Java heap size was set to 64 MB, with a maximum of 256 MB.

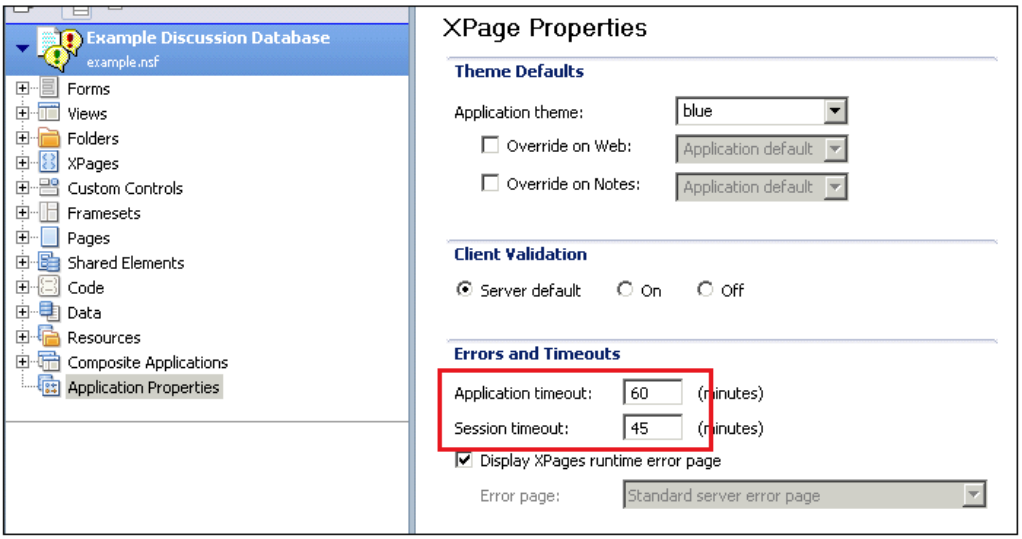
The Domino Administrator should work with the Domino Developer to determine a suitable Java heap size. The appropriate setting most likely depends on your use of XPages:

- If you are not using XPages applications at this time, 256 MB is too much memory to allocate to the Java heap. The HTTPJVMMaxHeapSize value should be set to 64M (64 MB).
- If you are exclusively using XPages applications, you may want this value set to 256 MB to support the additional demands on memory that XPages require. To prevent Domino from resetting the value back to 64 MB (prior to Domino 8.5.2), an additional notes.ini parameter is required:

  ```
  HTTPJVMMaxHeapSizeSet=1
  ```

6.6.2 Scoped variables

As described in 4.10 Scoped variables, heavy use of scoped variables in an XPage application can affect the memory usage (and, thus, performance) of your Domino server. If using sessionScope and applicationScope variables, the default time out period is around 30 minutes. You can increase or decrease these amounts through application settings:



Increases to these values mean that the variables will remain in memory for longer periods of time, which could have a negative impact on performance.

The down side of a short timeout is that the user may be annoyed because they were in the middle of something, went to lunch or whatever, and the server logged them out. Gauge the appropriate value for this parameter based on your understanding of how people will use the application, expected server load, and total number of users of the application expected in a given period of time. For instance, a website for use by the general public, with thousands of subscribers, may need a short timeout compared to, say, a call center application where the same 100 people are using the application all day long.

6.6.3 Do not allow simple search

Code that does full text searches to build collections should be run against applications that have a full text index built and maintained by the Domino server. There is an application setting called “Don't allow simple search” that prevents a full text search from being performed against an application that does not have a full text index. This means that if you have a scheduled agent that does a full text search, and this property is enabled, your scheduled agent will fail. Without the property, the agent would build a temporary full text index in memory, search that index, and continue processing. This is very inefficient and can be very resource intensive on the Domino server. You should make sure that any time a full text search is going to be used in an application, there is a full text index created for that application.

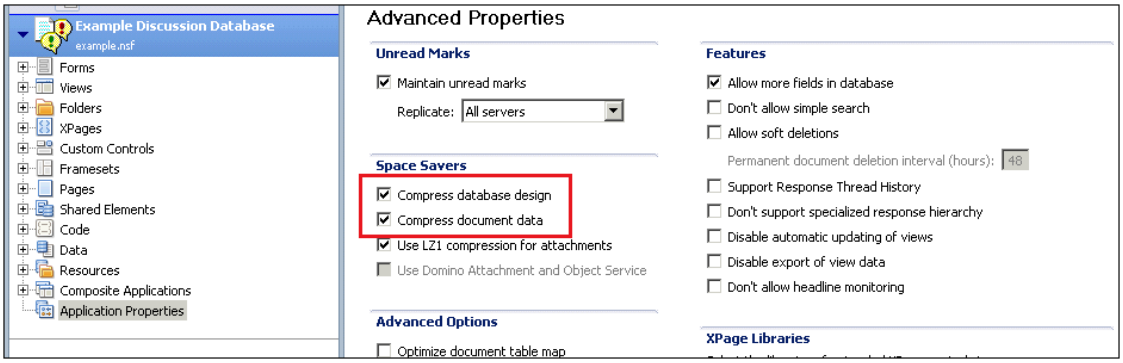
The setting also affects users who may do searches in your application. In the Notes client, any user can open a search bar in a view and try to do a full-text search. If there is

no full-text index, these user-initiated searches can be very resource intensive, so enabling this option prevents user actions from tying up the server.

6.6.4 Design and document compression

In order to reduce the disk spaced used by your application, you should enable the application property “compress database design”. This feature requires that the on disk structure (ODS) of the application be ODS 48 or later. Once the setting is enabled, you should work with your administrator to schedule a copy-style compact of the application to compress the design of existing design elements.

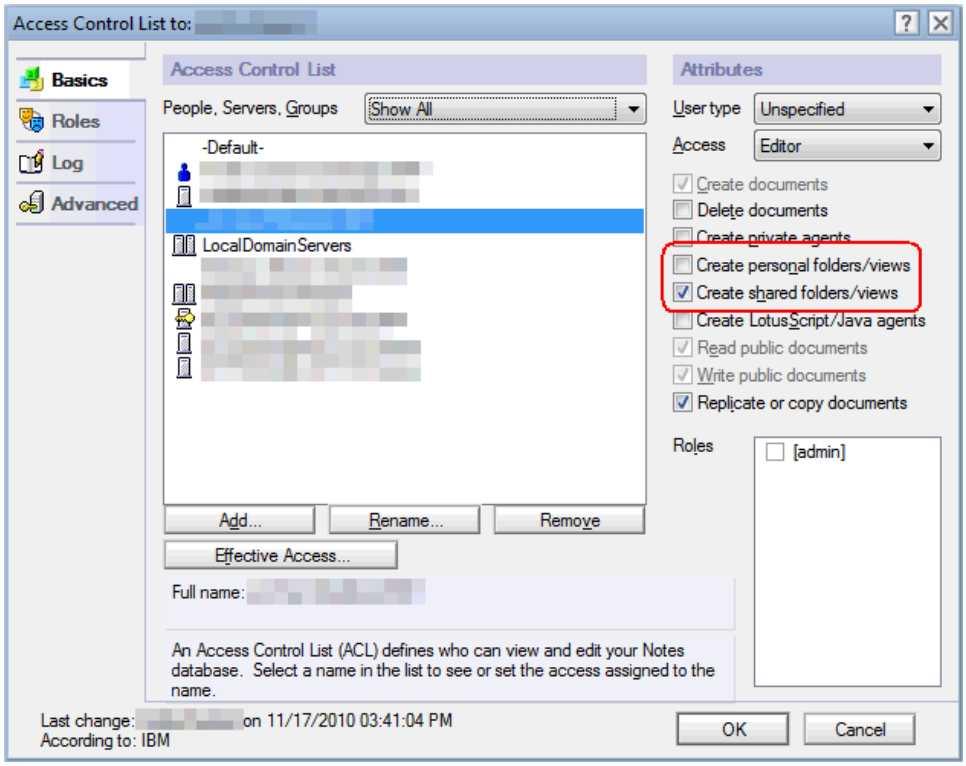
Furthermore, if your application contains large rich text fields, then enabling document compression is also a best practice. If your application has little or no rich text data, then it is best to not enable document compression. Again, the application must be at ODS 48 or higher, and a copy-style compact is required to compress existing document data.



6.6.5 Attaching large files with a browser

If you are designing a web application where potentially large file attachments are to be added to documents through a browser, you should work with your administrator. Attaching large files using web browser can create out of memory conditions on the Domino server. There is a notes.ini entry called DominoMultiPartPostBufferSize that can be used to control the size of the memory buffer used to process multi-part form data. You should work with your administrator to determine the appropriate value for this notes.ini entry.

6.6.6 User creation of views



As shown above, two ACL options can grant users access to create personal views on the server, or to create shared views on the server.

If you give ordinary end users these accesses, they may cause performance issues on your server by creating many views that will cause server slowdowns because of all the resources allocated to keeping the view indexes up to date. In addition, there's a potential security exposure that you should be conscious of.

If there's a legitimate need for a view, it's generally better for the end user to work with a developer to make sure the view fits into the UI scheme of the application, doesn't duplicate existing functionality, and is designed to minimally accomplish the needed task (e.g. that it has just the columns needed, not a column for every field on the form). Only trusted users who know what they're doing should be granted access to create shared views. Otherwise, you may see a proliferation of overly complex views that make a lot of work for the UPDATE task and cause slowdown of the server generally.

The performance penalty for "server private" views (the first of the two checkboxes) is less of a concern than for shared views. The indexes of server private views are not maintained by the UPDATE task; they are only refreshed when the user actually uses them. However, depending on the complexity of the view design, the number of documents in the view, and the amount of turnover, that can still be a fair amount of work for the server to execute on demand.

In terms of security exposure, we don't want to explain in detail, to avoid giving people instructions how to exploit this. Instead we will just state the principal: in applications that use Readers fields for document-level security, do not allow end users either of these

accesses.

NOTE: Any user with a Notes client and Reader access to a database can also create "desktop private" views stored on their workstation. This may also make extra work for the server when the views are used, since the client will request summary information about all recently modified documents. However, it's not as much work for the server, and there's no setting to affect this anyway, so don't worry about it.

6.7. Graphics

Graphics on your application's pages, when done incorrectly, can really affect the perceived performance of your application by significantly increasing bandwidth requirements. Here are some of the best practices for dealing with graphics on your web pages.

6.7.1 Image types

When working with images, you should limit yourself to the three major file types: PNG, JPEG, and GIF. You should not use older formats such as BMP that do not compress very well.

GIF

GIF stands for Graphics Interchange Format, and is the only format that allows for animated images. It allows an image to reference a palette of up to 256 distinct colors. Because of the color limitation, the GIF format is not the best choice for color photographs. This format is best used for graphics, logos, or other images with solid colors.

JPEG

JPEG stands for Joint Photographic Experts Group (the group that created the standard). This type is most often used for photographs, which is why the type is used by most digital cameras. It is best suited for images that have smooth variation of tone and color, which is why it is best for photographs.

PNG

PNG stands for Portable Network Graphics, and was developed as a replacement for the GIF format. However, PNG does not support multiple images (animation). It supports a wider array of colors than the GIF format does. Many times, an original GIF image stores in a smaller file when stored as a PNG image. If bandwidth is a concern, you should use PNG instead of GIF for your logos, graphics, or other non-photographic images.

6.7.2 Caching

There are two types of caching when it comes to images – caching by the Domino server, and caching by the local browser or client. If a URL used to locate an image remains constant, it can be cached by the local browser or client and used on subsequent page requests by the same browser.

To take advantage of Domino server image caching, use Image Resources for your images. You can reference the Image Resource in your XPage designs. These images are cached by the Domino server and improve the performance for subsequent renderings of the page by different users.

6.7.3 Scaling

If you want to have an image and a thumbnail of the image, use two distinct images instead of trying to scale on the browser. There are no-charge Internet sites where you can convert an image to a thumbnail. The thumbnail does not need to have the same fidelity as the original image, and can therefore be a far smaller file size. If you scale on the browser, you are affecting the performance of your application in two ways:

- The full image has to be sent to the browser, increasing bandwidth over the thumbnail.
- The browser has to do the work of scaling the image down to the right size, which takes CPU cycles on the user's computer to complete.

Using two images does not take advantage of caching on the browser, but the disadvantages of increasing the bandwidth and the browser scaling more than offsets the caching advantage (which is not an advantage when an individual browser first visits a page).

6.7.4 Compression

There are two types of image compression: "lossy” and “lossless". When an image is compressed, if done using a "lossless" compression, all the data in the original image is maintained. In a "lossy" compression, some of the data is removed from the original file to achieve a smaller file size. Depending on the image, the changes to the image may be imperceptible. Photographs generally benefit the most from “lossy” compression. There are many web sites on the Internet that compress images that you upload to their server. If you are using graphics in your application, see if you (or a co-worker) notice the difference between the original and the compressed image. If the differences are difficult to distinguish, then you should use the compressed image with the reduced file size in your application – the reduced network traffic will improve the performance of your application.